

---

# **DataScience.com Platform Documentation Documentation**

***Release 4.2.1***

**DataScience.com Team**

**Mar 28, 2018**



<b>1</b>	<b>Account Setup</b>	<b>3</b>
1.1	Git Configuration	3
1.1.1	Managing Git Provider Connections	3
1.1.2	GitHub Authentication	4
1.1.3	Bitbucket Authentication	4
1.1.4	GitLab Authentication	6
1.1.4.1	GitLab Enterprise v9 and Higher	6
1.1.4.2	Gitlab Enterprise v7 and v8	7
1.2	Kerberos Authentication	7
1.3	MapR Authentication	7
1.3.1	MapR Ticketing	7
1.3.2	MapR Username and Password	8
1.4	Collaborators and Permissions	8
1.4.1	User Permissions	8
1.4.2	Project Permissions	8
1.4.3	Teams	9
1.5	Environment Variables	10
1.5.1	Global environment variables	10
1.5.2	Project environment variables	10
1.6	Platform Configuration	11
1.6.1	Introduction	11
1.6.2	Instance Footprint	11
1.6.2.1	Production	11
1.6.2.1.1	3 Master Nodes	11
1.6.2.1.2	2 Postgres Nodes	12
1.6.2.1.3	Worker Nodes	12
1.6.3	Host Requirements	12
1.6.4	Supported Operating Systems	13
1.6.4.1	.deb Distributions	13
1.6.4.2	.rpm Distributions	13
1.6.5	Supported Browsers	13
1.6.6	Additional Software	13
1.6.7	Email Integration	13
1.6.8	Port Configuration	13
1.6.9	Git Providers	14
1.6.10	Limits	15

1.6.11	Optional Supported Integrations . . . . .	15
<b>2</b>	<b>Version Control</b>	<b>17</b>
2.1	Version Control . . . . .	17
2.1.1	File Previews . . . . .	17
2.1.2	Git Actions in the Platform . . . . .	18
<b>3</b>	<b>Projects</b>	<b>21</b>
3.1	Create a Project . . . . .	21
3.1.1	Create a project . . . . .	21
3.2	Navigation . . . . .	22
3.2.1	Navigating projects . . . . .	22
3.3	Project Collaborators . . . . .	22
3.4	Environment Variables . . . . .	23
3.4.1	Global environment variables . . . . .	23
3.4.2	Project environment variables . . . . .	23
3.5	Best Practices: Migrating Existing Work . . . . .	23
3.5.1	Moving Existing GitHub/GitLab/Bitbucket Repositories on the Platform . . . . .	24
3.5.2	Copying Files from Your Local Environment into a Project . . . . .	24
3.5.3	Migrating Work That is Not in a Version-Controlled Repository . . . . .	25
3.5.4	Warnings . . . . .	25
3.5.5	References . . . . .	25
<b>4</b>	<b>Environments</b>	<b>29</b>
4.1	Environments and Dependencies . . . . .	29
4.1.1	Introduction . . . . .	29
4.1.2	Browsing environments . . . . .	29
4.1.2.1	List page . . . . .	29
4.1.2.2	Details page . . . . .	29
4.1.3	Launching environments . . . . .	31
4.1.4	Adding additional requirements . . . . .	31
4.2	Environment Management . . . . .	32
4.2.1	Introduction . . . . .	32
4.2.2	Background . . . . .	32
4.2.3	What is an Environment? . . . . .	32
4.2.3.1	Base Environments . . . . .	33
4.2.3.1.1	Default Base Environment . . . . .	33
4.2.3.1.2	Customized Base Environments . . . . .	33
4.2.3.2	User Environments . . . . .	33
4.2.4	How to Create Environments . . . . .	33
4.2.4.1	Before You Begin . . . . .	33
4.2.4.2	Default Base . . . . .	34
4.2.4.3	Custom Base . . . . .	34
4.2.4.4	Create a User Environment . . . . .	37
4.3	Compute Resources . . . . .	40
4.3.1	On-demand Compute Resources for VPC Installations . . . . .	42
4.3.2	Tag Management for On-Demand Resources . . . . .	42
4.4	Using Spark on the Platform . . . . .	43
4.4.1	Jupyter . . . . .	43
4.4.2	RStudio . . . . .	44
4.4.3	Spark Usage . . . . .	44
4.4.3.1	Sparkmagic . . . . .	44
4.4.3.2	Spark in RStudio . . . . .	44
4.5	Best Practices: Choosing the Right Container Size . . . . .	44



4.5.1	Strategic Resource Allocation	45
4.5.2	Code Best Practices	45
4.6	Best Practices: Using Dependency Files	46
4.6.1	What are Dependency Files?	46
4.6.2	How to Create Dependency Files in a Jupyter Session	46
4.6.2.1	Creating a pip Dependency File in a Jupyter Python Session	46
4.6.2.2	Creating a Dependency File in an R Jupyter Session	46
4.6.2.3	Apt Dependency Files	46
4.6.3	Using Dependency Files on the Platform	49
4.6.3.1	In a Jupyter or RStudio Session	49
4.6.3.2	When Deploying an API	49
4.6.3.3	When Scheduling a Run	49
4.6.4	General Tips and Best Practices	49
4.6.5	Additional References on Dependency Files	52
<b>5</b>	<b>Working in a Session</b>	<b>53</b>
5.1	Launch a session	53
5.2	Sync changes	54
5.2.1	Git commands behind the scenes	56
5.3	Shut down a session	56
<b>6</b>	<b>Connect to Data Sources</b>	<b>57</b>
6.1	AWS Redshift	57
6.1.1	Python	57
6.1.1.1	Usage Example	58
6.1.2	R	58
6.1.2.1	Usage Example	59
6.2	AWS S3	59
6.2.1	Python	59
6.2.1.1	Usage Example	62
6.2.2	R	62
6.2.2.1	Usage Example	65
6.3	MySQL	65
6.3.1	Python	65
6.3.1.1	Usage Example	67
6.3.2	R	67
6.3.2.1	Usage Example	68
6.4	Google BigQuery	69
6.4.1	Python	69
6.4.1.1	Usage Example	70
6.4.2	R	70
6.4.2.1	Usage Example	71
6.5	SAP-HANA	71
<b>7</b>	<b>Scripts and Scheduled Runs</b>	<b>73</b>
7.1	Run a script	73
7.2	The Run Details page	73
7.3	Schedule a run	73
7.3.1	Custom schedules	76
7.4	Schedule Details page	76
<b>8</b>	<b>Reports</b>	<b>79</b>
8.1	Publish a Report	79
8.1.1	Preparing an .rmd File for Publishing	79
8.2	View and Manage a Report	81

8.3	Report Versions . . . . .	81
8.3.1	Publishing a New Version from Within a Report . . . . .	81
8.3.2	Publishing a New Version from the Action Button . . . . .	81
8.4	Delete a version . . . . .	85
<b>9</b>	<b>R Shiny Dashboards</b>	<b>87</b>
9.1	Publish a dashboard . . . . .	87
9.1.1	Running a Directory . . . . .	87
9.2	View and manage a dashboard . . . . .	88
<b>10</b>	<b>Deploy APIs</b>	<b>91</b>
10.1	Overview . . . . .	91
10.1.1	Deploy an API . . . . .	91
10.1.2	Call an API . . . . .	93
10.1.3	Manage an API . . . . .	93
10.2	Best Practices: Deploying an API . . . . .	93
10.2.1	Building the API Script . . . . .	94
10.2.1.1	The Deploy Timeout . . . . .	94
10.2.1.2	Pickling vs. Training . . . . .	94
10.2.1.3	Choosing a Response Type for a Deployed API . . . . .	95
10.2.2	Deploying the API . . . . .	96
10.2.2.1	Document Your Model or Function with a README . . . . .	96
10.2.3	Submitting Requests to Your API . . . . .	96
10.2.3.1	Send More Records and Fewer Requests . . . . .	98
10.2.3.2	Run APIs on Larger Containers to Improve Response Times . . . . .	98
10.2.3.3	Use Resource Pool over On-Demand for Faster Builds . . . . .	98
10.2.3.4	Prototyping to Production: Developing Internal Standards . . . . .	99
10.2.4	Dependencies . . . . .	99
10.2.4.1	Python Libraries . . . . .	99
10.2.4.2	R Libraries . . . . .	99
10.2.4.3	APT . . . . .	99
10.2.5	Examples . . . . .	100
10.2.5.1	Model . . . . .	100
10.2.5.2	Client . . . . .	100
<b>11</b>	<b>Appendix</b>	<b>103</b>
11.1	Dockerfile Basics and Best Practices . . . . .	103
11.1.1	Best Practices . . . . .	103
11.1.2	Dockerfile Basics . . . . .	104
11.1.2.1	Dockerfile Supported Instructions . . . . .	104
11.1.2.1.1	RUN Command . . . . .	104
11.1.2.1.2	SHELL Instruction . . . . .	104
11.1.2.1.3	COPY Instruction . . . . .	104
11.1.2.1.4	ADD Instruction . . . . .	104
11.1.2.1.5	ENV Instruction . . . . .	105
11.1.2.1.6	USER Instruction . . . . .	105
11.1.2.2	Instructions Not Allowed . . . . .	105
11.1.2.2.1	ARG Instruction . . . . .	105
11.1.2.2.2	FROM Instruction . . . . .	105
11.1.2.2.3	CMD Instruction . . . . .	105
11.1.2.2.4	ENTRYPOINT Instruction . . . . .	105
11.1.2.2.5	EXPOSE Instruction . . . . .	105
11.1.2.2.6	VOLUME Instruction . . . . .	106
11.1.3	Putting It All Together . . . . .	106

11.1.3.1	Example 1: Building a Conda Python 2.7 environment with ML and Stats Dependencies	106
11.1.3.1.1	A Conda Base Dockerfile:	106
11.1.3.1.2	Example User Environment Dockerfile:	106
11.1.3.2	Example 2: Installing R dependencies (rJava)	106
11.1.3.2.1	A Base Dockerfile for rJava Dependencies:	106
11.2	Enabling Hadoop and Spark	108
11.2.1	Introduction	108
11.2.2	Hadoop Cluster Configuration	108
11.2.2.1	Enabling Hadoop (Optional)	109
11.2.2.1.1	Optional Files	109
11.2.2.2	Enabling Hive (Optional)	109
11.2.2.2.1	Required Files	109
11.2.2.2.2	Optional Files	109
11.2.2.2.3	Tez	110
11.2.2.3	Enabling Spark	110
11.2.2.3.1	Required Files	110
11.2.3	MapR	110
11.2.3.1	Build a MapR Environment	110
11.2.3.2	MapR Ticketing	111
11.2.4	Other Providers	111
11.3	Git Provider Integration	111
11.3.1	Introduction	111
11.3.2	Supported Providers	112
11.3.3	GitHub OAuth Integration	112
11.3.3.1	Create a GitHub OAuth Application	112
11.3.3.2	Connect to Your GitHub OAuth Application	112
11.3.4	Bitbucket Integration	113
11.3.5	GitLab Integration	113
11.3.6	Manually Editing Providers in Postgres	113
<b>12</b>	<b>Release Notes</b>	<b>115</b>
12.1	Version 4.2.2 - October 4, 2017	115
12.1.1	Features	115
12.1.1.1	Select files to sync	115
12.1.1.2	Resource Management for Users	115
12.2	Version 4.1.1 - September 20, 2017	115
12.2.1	Features	115
12.2.1.1	Built-in MapR Hadoop support for Hive and Spark	115
12.3	Version 4.0.1 - September 6, 2017	116
12.3.1	Features	116
12.3.1.1	Environment Management	116
12.3.1.2	Sync and Shutdown from Jupyter sessions	116
12.3.1.3	File path autocomplete	116
12.3.1.4	Enhanced Platform availability	116
12.3.1.5	Single Sign On with SAML 2.0	116
12.4	Version 3.9.1 - August 23, 2017	116
12.4.1	Features	116
12.4.1.1	Report versioning	116
12.4.1.2	User-supplied custom tagging for Amazon EC2 on-demand resources	116
12.5	Version 3.8.1 - August 9, 2017	117
12.5.1	Enhancements	117
12.6	Version 3.7.1 - July 26, 2017	117
12.6.1	Features	117

12.6.1.1	R Shiny dashboards deployable to the Outputs page	117
12.6.1.2	Resource Management Dashboard	117
12.7	Version 3.6.1 - July 13, 2017	117
12.7.1	Features	117
12.7.1.1	H2O.ai dependency collection	117
12.7.2	Enhancements	117
12.7.2.1	Enhanced support for Internet Explorer 11	117
12.7.2.2	Shortcut links for returning to interactive sessions in progress	118
12.8	Version 3.5.1 - June 28, 2017	118
12.8.1	Features	118
12.8.1.1	Administrator-configured compute resources sizes	118
12.8.1.2	Various user experience and usability enhancements	118
12.9	Version 3.4.1 - June 15, 2017	118
12.9.1	Features	118
12.9.1.1	Curated dependency collections	118
12.9.1.2	Multiple language kernels available in Jupyter sessions	118
12.10	Version 3.3.1 - June 7, 2017	118
12.10.1	Features	119
12.10.1.1	GitHub Enterprise and GitLab Enterprise integrations	119
12.10.1.2	Global Environment Variables	119
12.10.1.3	On-demand compute resources in AWS VPCs	119
12.10.1.4	LDAP	119
12.11	Version 3.2.1 - May 31, 2017	119
12.11.1	Features	119
12.11.1.1	Bitbucket.org and GitLab.com integrations	119
12.11.1.2	RStudio	119
12.11.1.3	Publish RMarkdown HTML docs	119
12.12	Version 3.1.1 - May 4, 2017	120
12.12.1	Features	120
12.12.1.1	Projects	120
12.12.1.2	GitHub Integration	120
12.12.1.3	Secret Management	120
12.12.1.4	Launch Jupyter Interactive Sessions	120
12.12.1.5	Publish Reports	120
12.12.1.6	Deploy APIs	120
12.12.1.7	Run Scripts	120
12.12.1.8	Schedule Runs	120
<b>13</b>	<b>Tutorials and Examples</b>	<b>121</b>
13.1	Learning Modules	121
13.1.1	Use Shiny on the DataScience.com Platform	121
13.1.2	Connect Tableau to Model APIs on the DataScience.com Platform	121
13.2	Examples	122
13.2.1	How to Create and Deploy a Shiny App	122
13.2.1.1	Loading the Data	122
13.2.1.2	Defining the UI Components	123
13.2.1.3	Defining the Server Component	123
13.2.1.4	Running the App	124
13.2.1.5	Publishing the App	124
13.2.2	Using a Deployed API	126
13.2.2.1	Business Use Case	126
13.2.2.2	Training the Model	126
13.2.2.3	Fitting the Model	129
13.2.2.4	Saving the Model	129

13.2.2.5	Deploying the Model . . . . .	129
13.2.2.6	Calling the API . . . . .	131
13.2.2.7	Saving the Output . . . . .	133
13.2.3	Deploying a Network Intrusion Prediction API . . . . .	133
13.2.4	Deploying an XGBoost Model . . . . .	136
13.2.4.1	The Business Use Case . . . . .	136
13.2.4.2	Loading the Data and Training the Model . . . . .	139
13.2.4.3	Deploying the Model . . . . .	140
13.2.4.4	Conclusion . . . . .	143
<b>14</b>	<b>How to Read These Docs</b>	<b>145</b>
<b>15</b>	<b>Just Getting Started with the Platform?</b>	<b>147</b>



The DataScience.com Platform combines the tools, libraries, and languages your team loves with the infrastructure and workflows your organization needs. The Platform combines three key components:

- **Infrastructure** - systems tasks (like spawning servers) are abstracted and handled automatically so data scientists can focus on the substance of their work
- **Tools** - open source tools (like Jupyter, R Shiny, or modeling libraries) that data scientists need are integrated into a centralized place
- **Workflow** - automation for tasks, collaboration, and communication that let data science teams effectively deliver on their mission





As an individual user, you'll set up your Git provider credentials and (optionally) authenticate with your Hadoop data lake. This section outlines steps to get those integrations connected.

### 1.1 Git Configuration

Every analysis that runs on the Platform, from Jupyter sessions to Model APIs, uses code stored in a Git repository (commonly referred to as a “repo”). Actions like runs, APIs, and sessions all start from a specific version (called a “commit”) of the repo, letting you maintain a transparent record of your process.

It's good to understand the basics of Git before using the DataScience.com Platform. [This page](#) of resources curated by GitHub is a good start.

---

**Note:** There is a one-to-one relationship between projects on the platform and Git repositories. Once you've connected a repo to a project, your team may not connect it to any other project. This rule keeps each project's analyses closely tied to changes in the source code so others can retrace your steps.

---

The Platform supports Git repositories hosted on GitHub (including GitHub Enterprise), Bitbucket (including Bitbucket Server), and GitLab (both Community and Enterprise). You'll use your credentials for these Git providers to work with repos directly in the DataScience.com Platform.

Before your team can authenticate with a Git provider, an Admin should follow the instructions in the [Git Provider Integration](#) docs.

#### 1.1.1 Managing Git Provider Connections

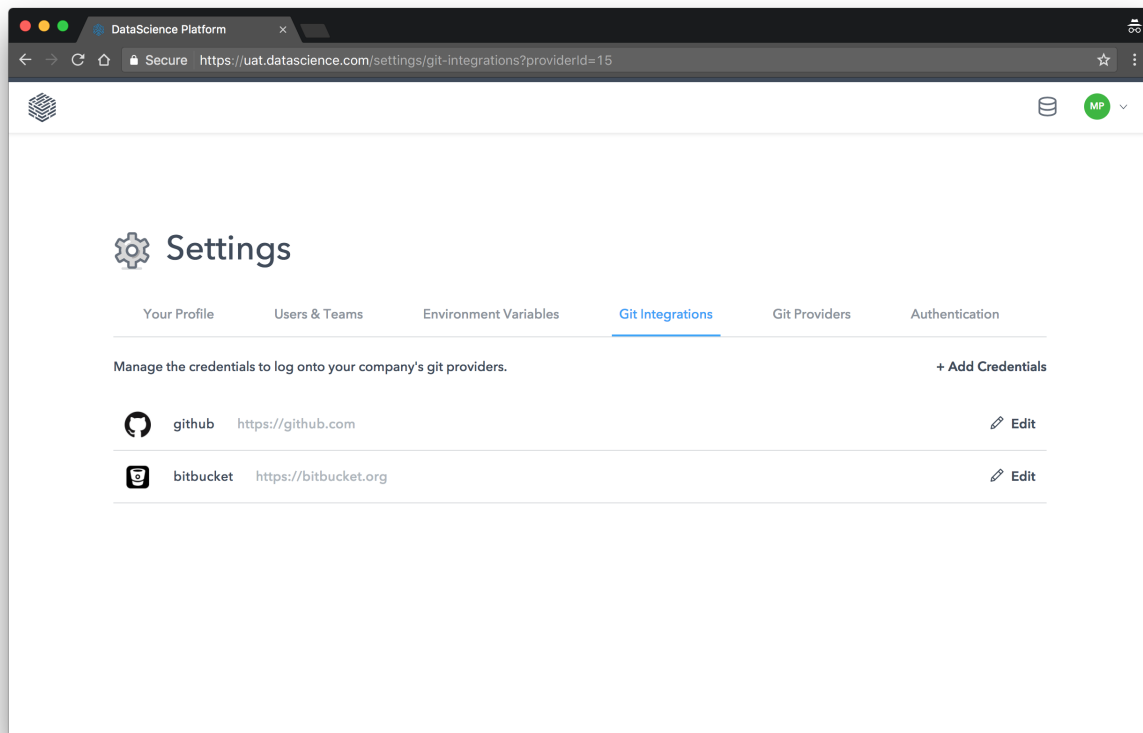
After an Admin has registered a Git provider with DataScience.com, you can connect your account with the following steps:

1. On the DataScience.com Platform, visit Settings > Git Integrations.
2. Click Add Credentials and search for providers registered by your Administrator.

3. Complete the form with your Git provider account information.

Your registered Git connections appear on this Git Integrations page. From this list, you can edit and remove connections.

Disconnecting from this menu means that the Platform will clear out your authentication information (tokens, passwords, etc.) from its secure storage. Removing your authentication won't cause anything to be lost on the Platform, but you will no longer be able to see files or launch analyses in your projects that were connected with that Git provider.



### 1.1.2 GitHub Authentication

The GitHub integration uses GitHub's OAuth features to connect your account. On the DataScience.com Platform, you'll only have to click the Connect button in the GitHub integration form to sign in and approve access.

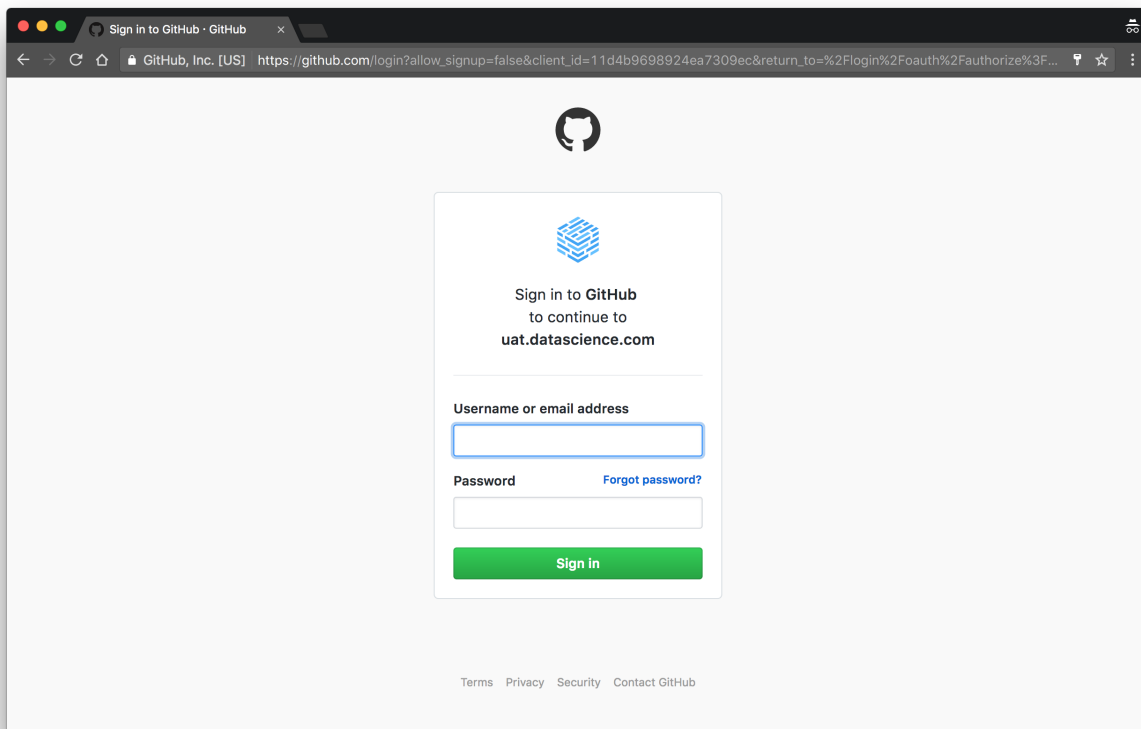
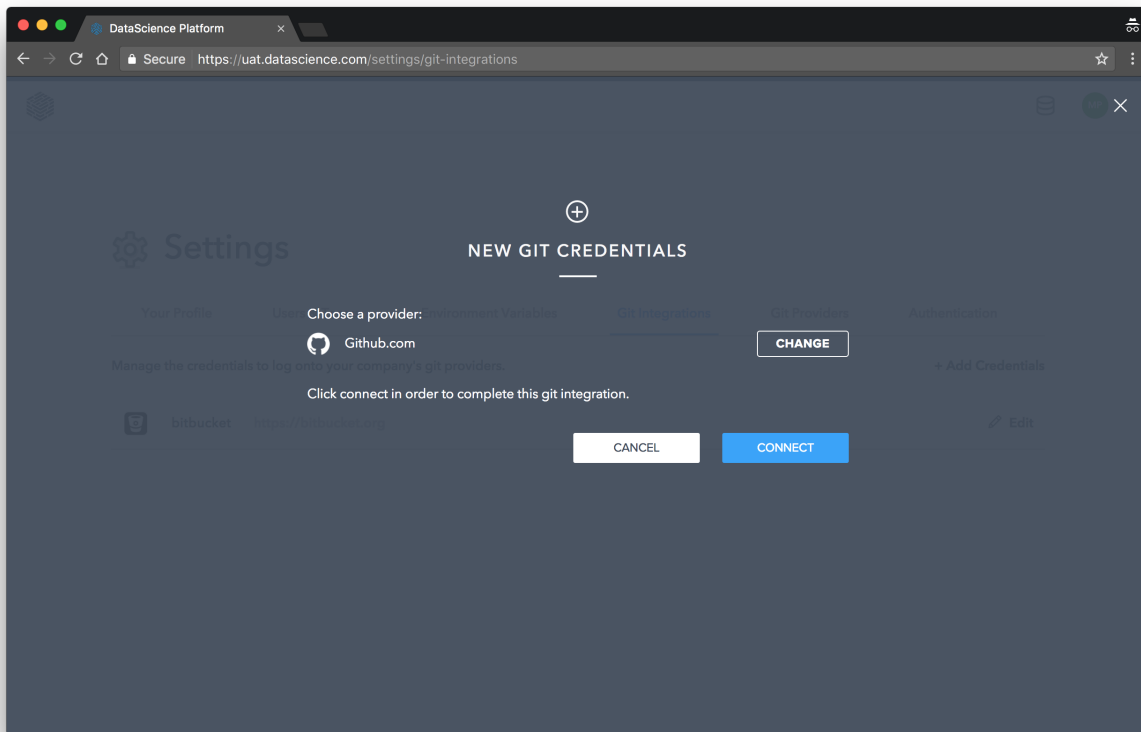
You'll be prompted to enter your GitHub username and password. Once you're successfully connected, you'll be redirected back to the Platform.

### 1.1.3 Bitbucket Authentication

The Bitbucket integration uses Bitbucket's App Passwords feature to grant repo access. A Bitbucket App Password is just like your account password but meant for other apps to control Bitbucket on your behalf.

Start by creating a Bitbucket App Password inside the Bitbucket web app. Visit Bitbucket Settings > App Passwords and select Create an App Password. Give the app password a name and select the following permissions:

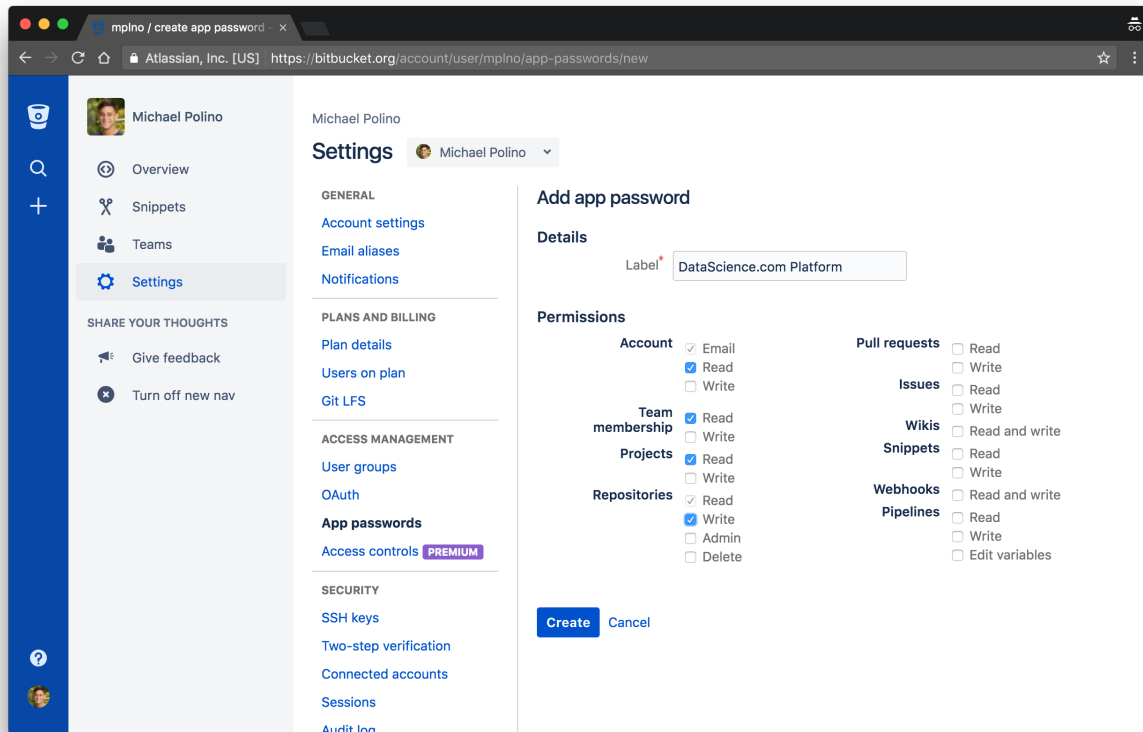
Account: Email + Read



Team membership: Read

Projects: Read

Repositories: Read + Write



After you create an App Password on Bitbucket and copy it from the pop-up window, you'll no longer be able to view the password on Bitbucket. Keep this password safe by immediately copying it over to the DataScience.com Platform.

With your Bitbucket App Password copied, go to the DataScience.com Platform and visit Settings > Git Integrations. Click Add Credentials, search for your Bitbucket integration, and enter your user name, email, and app password.

You can find your Bitbucket username in the Bitbucket app by visiting your account settings. For more information on Bitbucket App Passwords, see [their docs](#).

## 1.1.4 GitLab Authentication

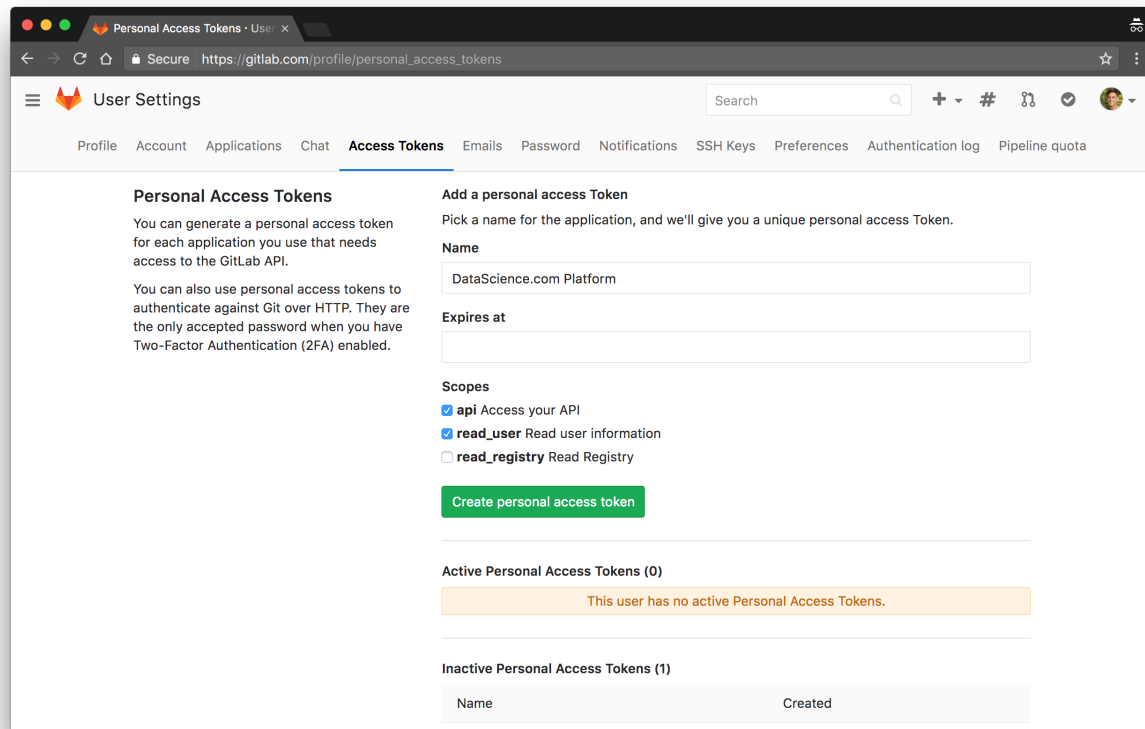
### 1.1.4.1 GitLab Enterprise v9 and Higher

The GitLab integration uses GitLab's Access Tokens feature to grant repo access. A GitLab Access Token is just like your account password but meant for other apps to control GitLab on your behalf.

Start by creating a GitLab Access Token inside the GitLab web app. Visit User Settings > Access Tokens. Give the app password a name, select `api` and `read_user` under Scopes, then click Create Personal Access Token.

After you create a token on GitLab and copy it from the confirmation screen, you'll no longer be able to view it on GitLab. Keep this token safe by immediately copying it over to the DataScience.com Platform.

With your GitLab token copied, go to the DataScience.com Platform and visit Settings > Git Integrations. Click Add Credentials, search for your GitLab integration, and enter your user name, email, and access token.



### 1.1.4.2 Gitlab Enterprise v7 and v8

These versions only require your GitLab username and password to allow access to the GitLab repo. Creating Access Tokens is not necessary.

## 1.2 Kerberos Authentication

With the Kerberos integration enabled, all of your analysis containers (including interactive sessions, script runs, and Model APIs) will automatically authenticate as your user-specific Kerberos principal.

To authenticate via Kerberos, visit Settings > Authentication, enter your username, and upload your own keytab file. With your keytab uploaded, each new analysis you launch will be authenticated through Kerberos.

## 1.3 MapR Authentication

Users have the option to authenticate to MapR via ticketing or username/password. With your MapR credentials stored securely on the Platform, each new analysis you launch will be authenticated with the cluster.

### 1.3.1 MapR Ticketing

The MapR Ticket integration lets you automatically authenticate with your MapR cluster in any analysis launched on the Platform. To configure the MapR Ticket integration, visit Settings > Authentication, then enter your MapR username and upload your own MapR Ticket.

## 1.3.2 MapR Username and Password

Alternatively, you can enter your MapR cluster username and password in Settings > Authentication.

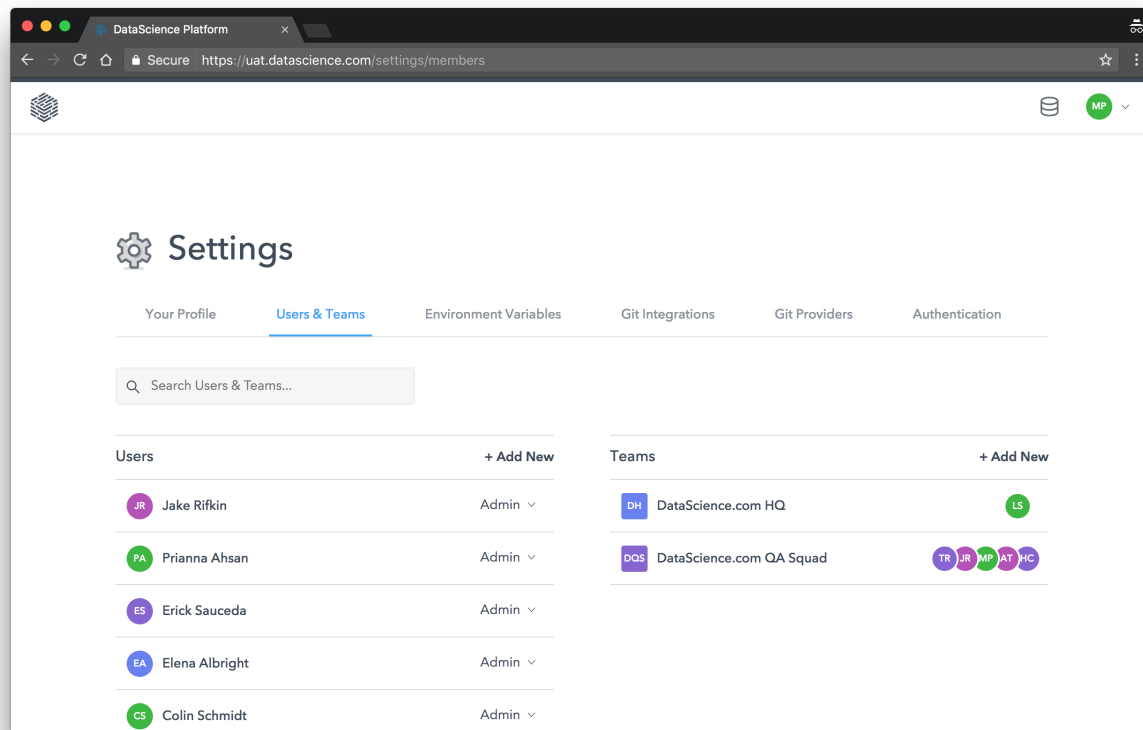
## 1.4 Collaborators and Permissions

### 1.4.1 User Permissions

There are three levels of permissions on the Platform:

- **Admin:** has access to all projects, can create other admins, and can shutdown/manage any user-created analysis processes (e.g. a Jupyter session)
- **Standard:** may be invited to projects as owners, editors, or viewers
- **Read-Only:** may only be invited to projects as viewers

If you're an Admin, you can add and remove users by visiting Settings > Users & Teams.



### 1.4.2 Project Permissions

By default, when you create a project you will be the only one who can see or change it. It will not appear in the All Projects list for anyone else. To give others access, visit the Project Settings tab and click Invite Collaborators.

When you add a collaborator to a project, you'll assign one of three permission levels. The following actions are available to each level:

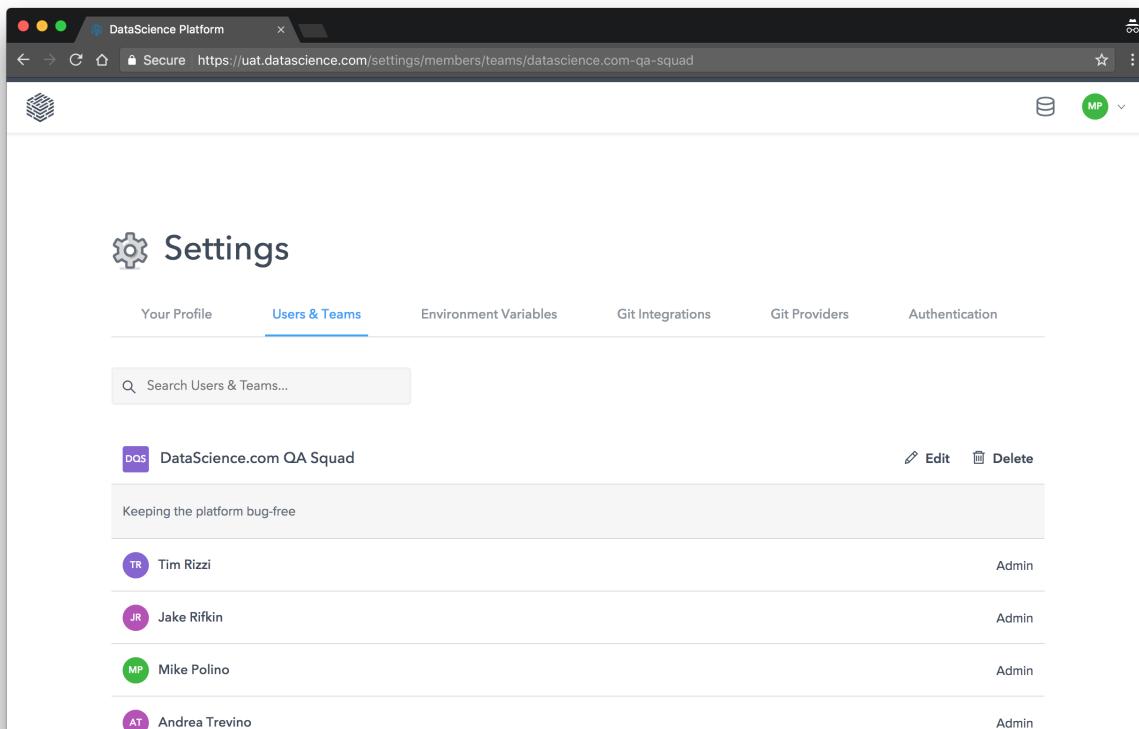
Action	Own	Edit	View
View outputs	Yes	Yes	Yes
View activity	Yes	Yes	Yes
View files	Yes	Yes	No
Launch containers (runs; sessions; APIs; etc.)	Yes	Yes	No
Edit environment variables	Yes	Yes	No
Invite collaborators with 'read' access	Yes	Yes	No
Invite collaborators with 'edit' access	Yes	Yes	No
Invite collaborators with 'own' access	Yes	No	No

**Warning:** Admin users have full access to all projects, regardless of whether they have been added as a collaborator. Read-Only users can only have View permission to a project, regardless of what permission they've been given by the project owner.

### 1.4.3 Teams

You can group teammates into teams and add them as collaborators to projects all at once with the same permission level. To create a team, visit Settings > Users & Teams and click Add New.

You can visit a team by clicking on the team name. From this page, you can add or remove members and edit the team info:



If you're added to a project as an individual and through a team invite, you'll get the most specific permission level first, followed by the highest permission level. For example, if you're given View permission to a project and you're

on a team with Edit permissions, you'll have View permission. If you're on two teams added to a project, one with View and the other with Edit permission, you'll have Edit permissions.

## 1.5 Environment Variables

You can configure environment variables both for projects and globally on the platform. These key-value pairs are kept in encrypted storage and injected into your analysis at runtime. The primary purpose of this feature is to help you avoid checking sensitive information (like a database password) into your Git repo.

You can use these environment variables like you would in any other analysis. For example, if you create an environment variable on the platform with key `REDSHIFT_PASSWORD` and value `im_secure`, you can run the following example in Python:

```
import os
print os.environ['REDSHIFT_PASSWORD']
>>> im_secure
```

There are a few rules to remember when using environment variables:

- Environment variables are injected once into your analysis when you launch it. You must shut down and re-launch any analyses to access new variables.
- If you need to edit an existing environment variable, you must delete it and create a new one.
- If a global environment variable and a project environment variable share the same key, the project environment variable will take precedence.

### 1.5.1 Global environment variables

Admins can create global environment variables and assign them to users or teams. To create a global environment variable, navigate to Settings > Environment Variables. Click Add New and fill in the key, value, description, and users that can access this variable.

**Warning:** If you don't assign a global environment variable to any users, it will be shared with all users on the Platform by default.

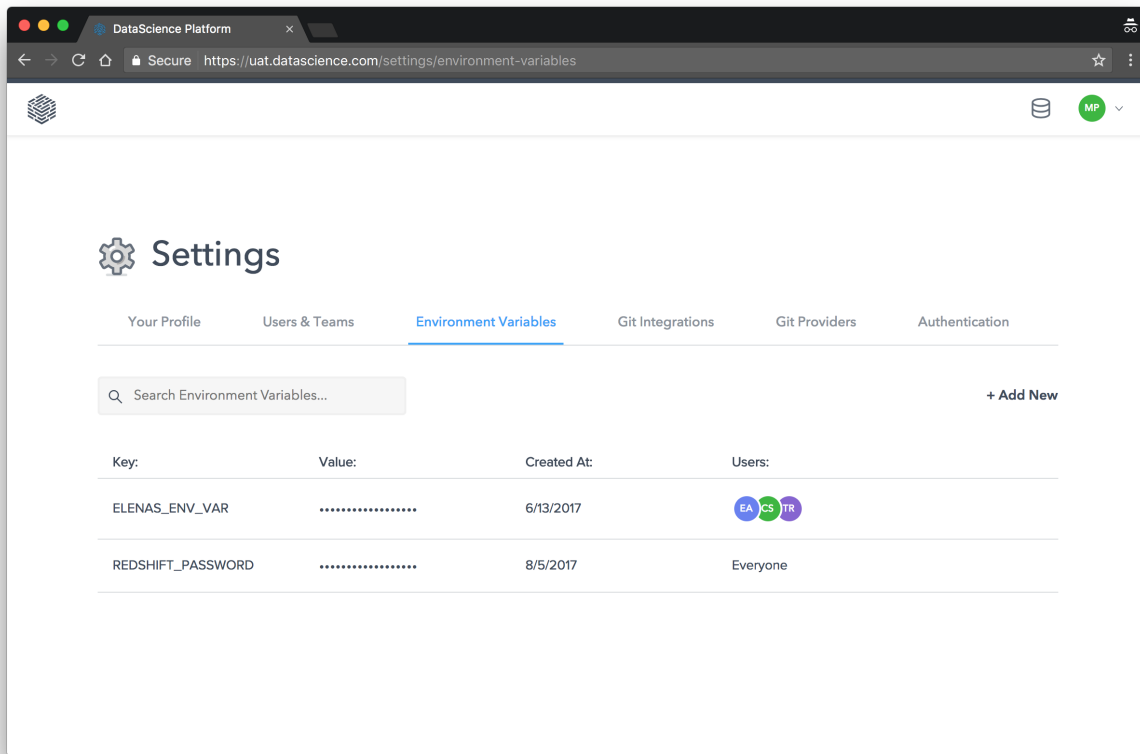
After creating a global environment variable, you can manage access to it by hovering over its row in the table and selecting the View Details button. From this page, click Edit to add new users or remove them with the Remove buttons in the table.

Any user with the standard role can see the list of global environment variables but cannot create or delete global environment variables.

### 1.5.2 Project environment variables

If you have Own or Edit permission, you can add and remove project environment variables. You can find project environment variables on the project Settings tab. Remember that if you have a global and project environment variable with the same name, only the global variable will be available in your analysis.





## 1.6 Platform Configuration

### 1.6.1 Introduction

This short guide provides a brief overview of the key Platform specs. These are useful to know for the Users of the Platform.

Each installation is unique and comes with its own constraints. Double check with your IT department for any differences that could exist between this guide and your custom installation.

### 1.6.2 Instance Footprint

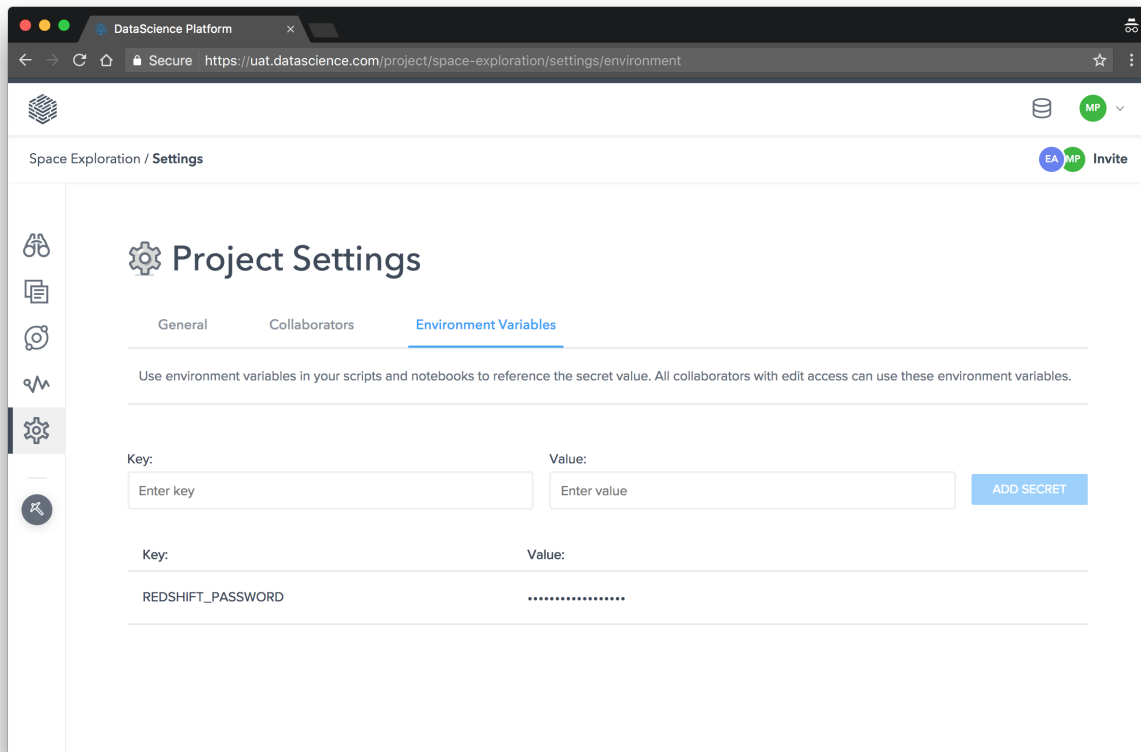
#### 1.6.2.1 Production

In a production instance, the DataScience.com Platform requires the following minimum number of servers or nodes. Note that it's crucial for the sake of both stability and security to separate the nodes that run the core Platform services from the nodes that host user workloads.

##### 1.6.2.1.1 3 Master Nodes

These run the core components of the Platform, split evenly across the three nodes in a leaderless cluster. In the case that one or more nodes fails, the remaining node(s) will take on the additional load of the failed Master.

These hosts should meet or exceed the minimum requirements in the Host Requirements section below.



### 1.6.2.1.2 2 Postgres Nodes

The two Postgres nodes consist of a master database and a stand-by database. In cloud installations, these can be one redundant managed database, such as what AWS RDS offers.

The hosts for these nodes should meet or exceed the requirements in the Databases section below.

### 1.6.2.1.3 Worker Nodes

User workloads are hosted on separate Worker nodes for increased stability and security. The number of Worker nodes and their sizing depends on the number of Platform users and how many resources you plan to allocate for each.

For instances of the Platform that are hosted on Amazon AWS, we also support on-demand instances for user workloads.

## 1.6.3 Host Requirements

Per host:

- RAM: 32GB
- CPU: 8 core
- Disk Space: 300GB

## 1.6.4 Supported Operating Systems

(64-bit distributions)

### 1.6.4.1 .deb Distributions

- Debian 9 (kernel 4.9+)
- Ubuntu 16.04 (kernel 4.4+)

### 1.6.4.2 .rpm Distributions

- Fedora 24 (kernel 4.11+)
- Red Hat Enterprise Linux 7.3 (kernel 3.10.0-514+)
- CentOS 7.3 (kernel 3.10.0-514+)

## 1.6.5 Supported Browsers

The DataScience.com Platform relies on native flexbox support, which requires the following minimum versions:

- Apple Safari 10+
- Google Chrome 49+
- Microsoft Edge 14+
- Microsoft Internet Explorer 11+ (partial support; there are some known issues with flexbox)
- Mozilla Firefox 51+
- Opera 43+

## 1.6.6 Additional Software

The installation script for the DataScience.com Platform will automatically install the correct version of docker-engine; please ensure this version is not overwritten by other configuration management tools.

- docker 17.06-ce+

## 1.6.7 Email Integration

The DataScience.com Platform requires an email server to send invitations and collaboration notifications. You will need an SMTP server address, port, username, password, and “From” address. Please see your System Administrator for more details.

## 1.6.8 Port Configuration

The following ports should be opened between the specified sources and destinations. “Administrative IP(s)” refers to the IP(s) from which Systems Administrators will need to access the instance. “User IP(s)” refers to the IP(s) from which users of the DataScience.com Platform will be accessing the application.

**Caution: LDAP and SMTP Ports**

For integrations such as LDAP and SMTP, we've provided the most commonly used ports. Please confirm these ports with your Service Administrator(s).

Port	Usage	Source(s)	Destination(s)
25	Unencrypted SMTP traffic	Master node	SMTP server
80	HTTP (redirects to HTTPS)	Administrative IP(s) & User IP(s)	Master node
389 (optional)	Non-SSL LDAP traffic	Master node	LDAP server
443	HTTPS	Administrative IP(s) & User IP(s)	Master node
465	Encrypted SMTP traffic	Master node	SMTP server
636 (optional)	SSL LDAP traffic	Master node	LDAP server
2376	Docker remote socket	Master node	All nodes
2377	Docker Swarm API	All nodes	All nodes
5000	Logstash ingress	All nodes	Master node
5432	Postgres traffic	Master & Core nodes	Postgres endpoint
7946	Docker Swarm	All nodes	All nodes
8080	HTTP (redirects to HTTPS)	Administrative IP(s) & User IP(s)	Master node
8085	GitHub OAuth authentication DS Docker Event Listener	github.com & All nodes	All nodes
8300-8302	Consul	All nodes	All nodes
8500	Consul	All nodes	All nodes
8600	Consul	All nodes	All nodes
8686	Darkroom	All nodes	All nodes
8800	Admin Console	Administrative IP(s)	Master node
8830	Acquiesce	All nodes	All nodes
8899	Graphite & Statsd	All nodes	Master node
9870 - 9880	Cluster management	All nodes	All nodes
32768-61000	Proxy routing to containers	Master node	All nodes

**Important: Connecting to data sources**

In addition to the above, please ensure that routes are open between the DataScience.com Platform and whatever data sources you plan to connect.

**1.6.9 Git Providers**

In order to create projects in the DataScience.com Platform, you must integrate with a Git provider. We currently support the following:

- GitHub.com

- GitHub Enterprise 2.9+
- Bitbucket.org
- GitLab.com
- GitLab Enterprise 7+

### 1.6.10 Limits

**File upload/download limit:** 200MB

**Google Compute Engine and SMTP:** GCE does not currently support the use of standard SMTP servers. They do, however, offer support for their own Gmail service as well as several third party providers, including [SendGrid](#).

### 1.6.11 Optional Supported Integrations

#### LDAP & Active Directory

If you use LDAP or Active Directory to manage users in your organization, your System Administrator can configure your DataScience.com Platform to use this integration to set up users and permissions. Optionally, when LDAP or Active Directory is enabled, you can also enable Single Sign-On.

#### On-Demand Compute Resources

If your DataScience.com Platform runs on Amazon AWS, your System Administrator can configure some or all of your services to run ad-hoc. This can save costs and resources.

#### 3rd Party Logging

The DataScience.com Platform currently has optional integrations with Loggly and Datadog. If you use either of these for logging or monitoring, your System Administrator can add your API keys to the Platform to send data to these services.



The Platform is integrated with third-party version control services like github or bitbucket. Learn how to use version control on the platform in this section

## 2.1 Version Control

### 2.1.1 File Previews

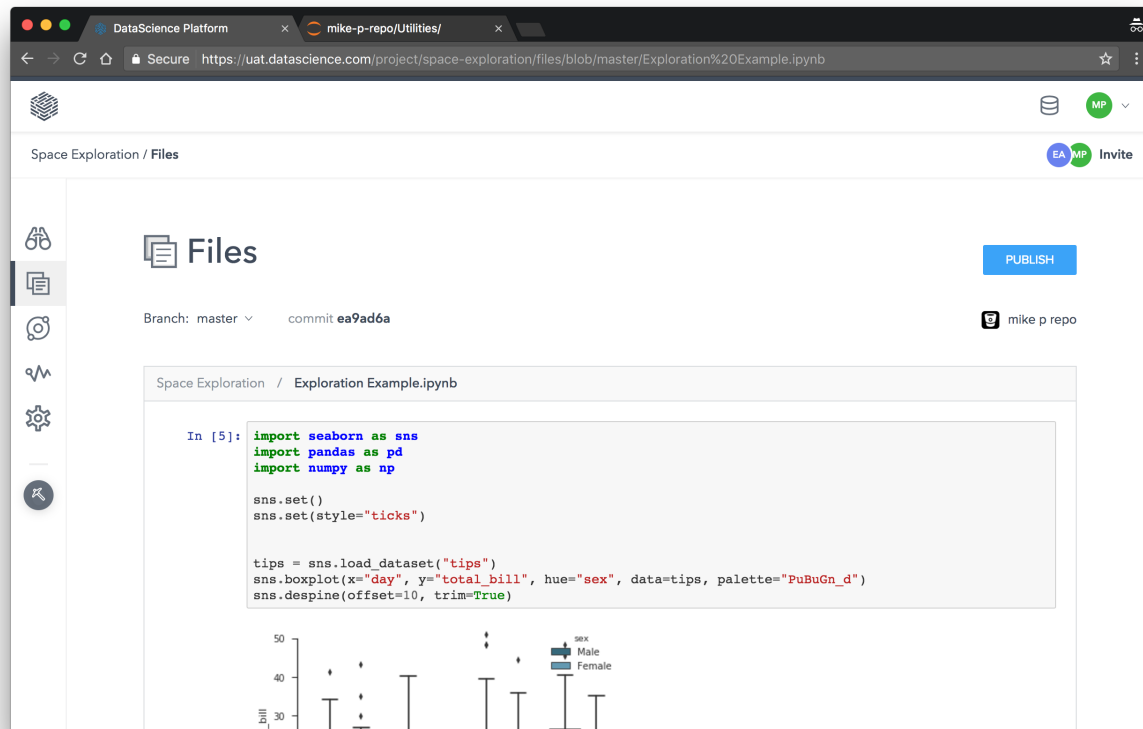
You can browse the files in a project's Git repo on the Files tab. On this tab, you can navigate through folders and preview files. Jupyter Notebooks (.ipynb), Markdown files (.md), and images (.jpg, .png, .gif) are rendered as HTML for easy previewing.

**Warning:** To ensure a fast and responsive app experience, data files (.csv, .tsv, .pkl, .hdf, .npy) and any other file larger than 5MB can't be previewed.

Files in a project repo are identified by their branch and commit. Above every list of files or file preview on the Files tab, you'll find a branch selector and a commit (represented by a short commit SHA). Any actions you take on the project, like publishing a report or launching an interactive session, will be associated to this branch and commit.

The Files tab is meant to be a lightweight way to preview files. To manage branches, explore the Git log, or take any other more advanced actions on your Git repo, use any of the following tools:

- [Git command line](#)
- [A Git GUI client](#)
- Tools included in the GitHub, Bitbucket, or GitLab web apps

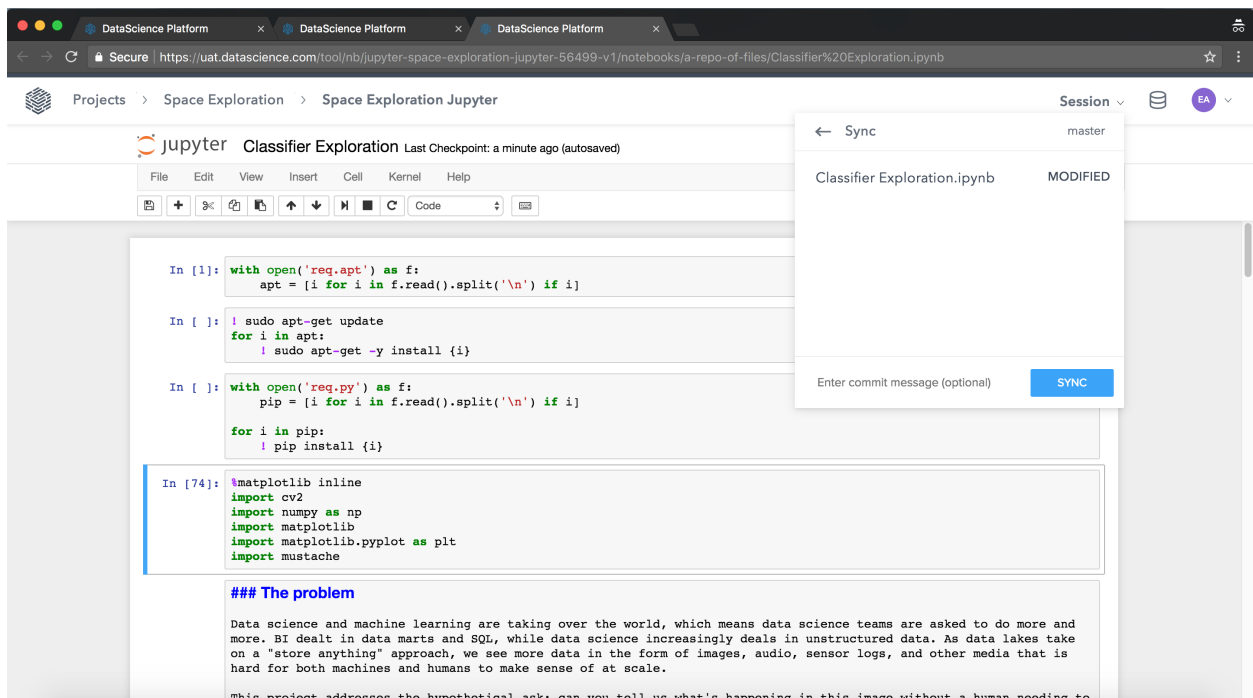


## 2.1.2 Git Actions in the Platform

Every process launched in the Platform, from Jupyter sessions to Model APIs, uses the code you select from your Git repo. When your analysis launches, the Docker container running it has Git configured with your credentials. From any running process, you could run commands `git status` just as you would on your own computer.

In interactive sessions (like Jupyter and RStudio), you don't have to run Git commands yourself. Instead, the session sync feature lets you save changes back to your Git remote from the browser without needing the Git command line interface. For detailed instructions on session sync, see the [Working in a Session](#) section.







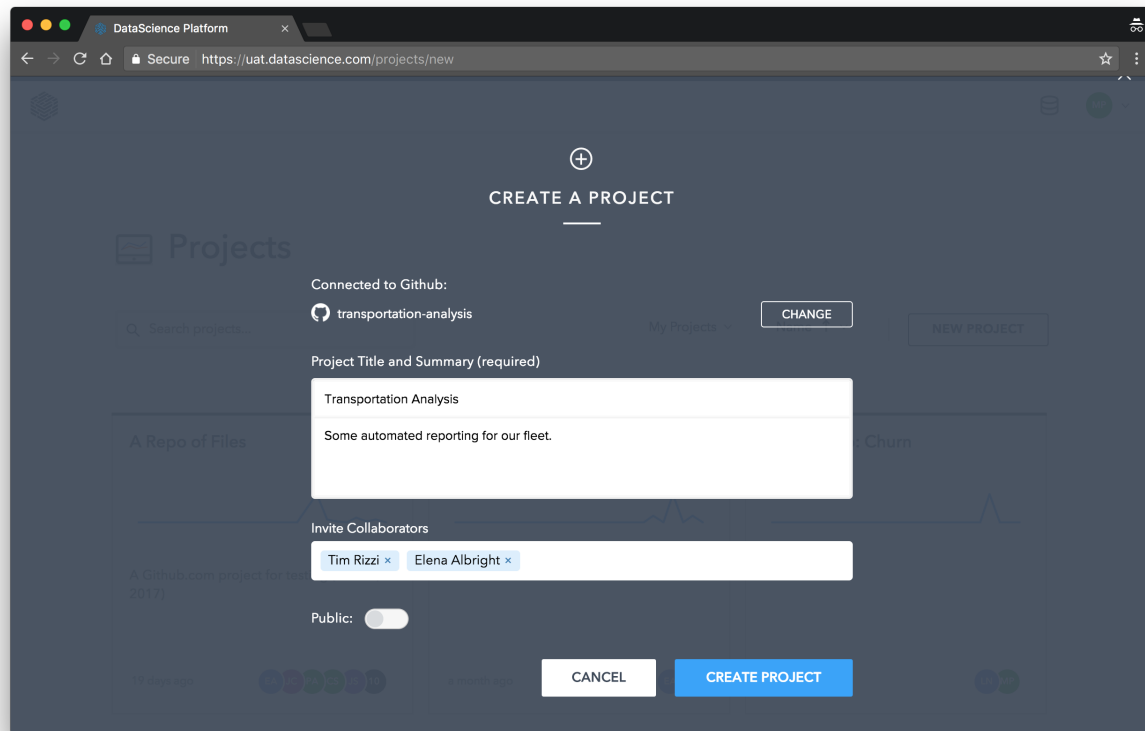
All work in the Platform is organized into projects. When you first log in, you will land on the All Projects page, which displays all projects in your instance that you have access to. You can return to this page from anywhere by clicking the DataScience.com logo in the top left corner of any page.

## 3.1 Create a Project

### 3.1.1 Create a project

To create a new project, click New Project in the top right of the All Projects page and follow these steps:

1. **Choose your Git provider** - Start by selecting the Git remote where you've stored your code. If you haven't already done so, connect to your Git provider before starting this step (see the [Git Configuration](#) docs for more).
2. **Select a repo** - Use the search bar to find and select the Git repo that will back your project. Remember that a repo may only back one project at a time.
3. **Write a name and description** - Your project must have a unique name (no longer than 55 characters) and description (no longer than 140 characters). Both can be edited later.
4. **Invite collaborators** - You may add teammates to your project and select their permission level (see [Collaborators and Access Control](#) for more).
5. **Choose public or private project** - If you select Public, all users of your instance will be able to view your project.
6. **Click "Create Project"**



## 3.2 Navigation

### 3.2.1 Navigating projects

Use the sidebar on the left of the screen to switch between the five main sections of every project:

- **Overview:** view a summary of your project, including its README, connected repo and a sparkline of activity
- **Files:** list and preview all files in your project
- **Activity** - see a log of completed, running, and scheduled jobs
- **Outputs** - access the collection of reports, APIs, and dashboards in your project
- **Settings** - view and edit project-level settings, like title, permissions, and collaborators

## 3.3 Project Collaborators

The Platform lets you control exactly who can see or edit your work. By default, when you create a project you will be the only one who can see or change it. It will not appear in the All Projects list for anyone else. To give others access, you must add them as collaborators in your project's settings.

When you add a collaborator, you can choose one of three permission levels: View, Edit, and Own. The list of available actions for each permission level is listed in the [Collaborators and Access Control](#) section.

## 3.4 Environment Variables

You can configure environment variables both for projects and globally on the platform. These key-value pairs are kept in encrypted storage and injected into your analysis at runtime. The primary purpose of this feature is to help you avoid checking sensitive information (like a database password) into your Git repo.

You can use these environment variables like you would in any other analysis. For example, if you create an environment variable on the platform with key `REDSHIFT_PASSWORD` and value `im_secure`, you can run the following example in Python:

```
import os
print os.environ['REDSHIFT_PASSWORD']
>>> im_secure
```

There are a few rules to remember when using environment variables:

- Environment variables are injected once into your analysis when you launch it. You must shut down and re-launch any analyses to access new variables.
- If you need to edit an existing environment variable, you must delete it and create a new one.
- If a global environment variable and a project environment variable share the same key, the project environment variable will take precedence.

### 3.4.1 Global environment variables

Admins can create global environment variables and assign them to users or teams. To create a global environment variable, navigate to Settings > Environment Variables. Click Add New and fill in the key, value, description, and users that can access this variable.

**Warning:** If you don't assign a global environment variable to any users, it will be shared with all users on the Platform by default.

After creating a global environment variable, you can manage access to it by hovering over its row in the table and selecting the View Details button. From this page, click Edit to add new users or remove them with the Remove buttons in the table.

Any user with the standard role can see the list of global environment variables but cannot create or delete global environment variables.

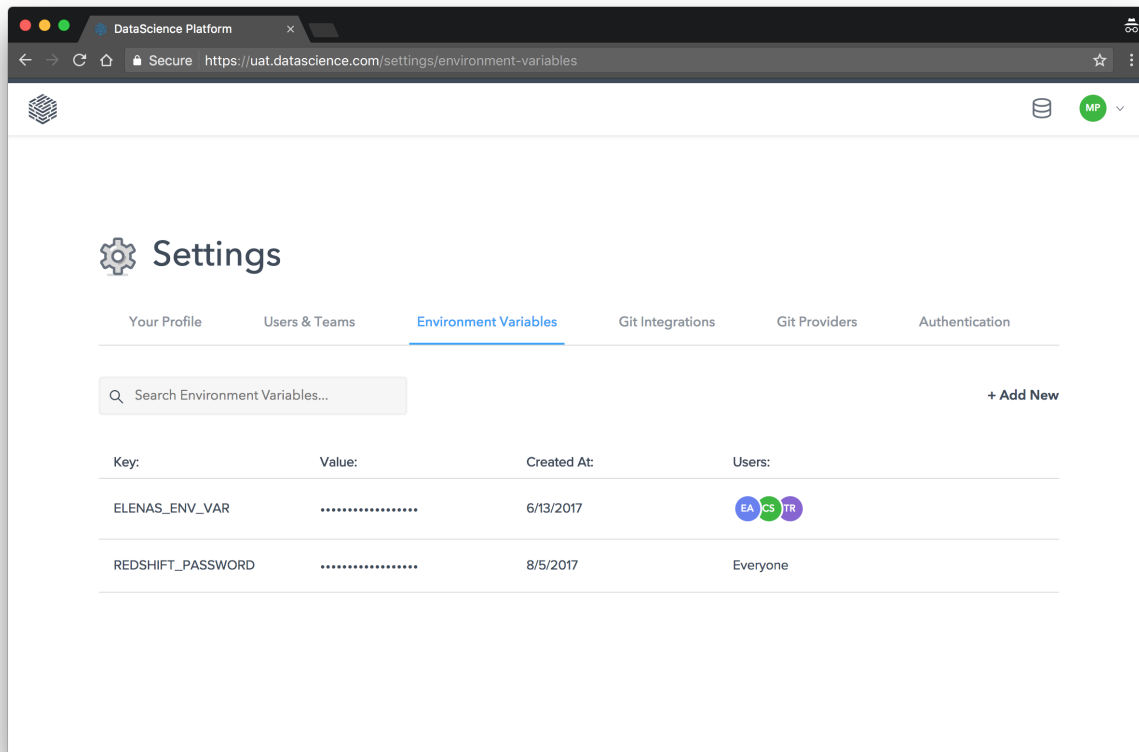
### 3.4.2 Project environment variables

If you have Own or Edit permission, you can add and remove project environment variables. You can find project environment variables on the project Settings tab. Remember that if you have a global and project environment variable with the same name, only the global variable will be available in your analysis.

## 3.5 Best Practices: Migrating Existing Work

In this article you will learn how to migrate existing work onto the Platform. There are three different example cases:

- Case 1: You want to move existing GitHub/GitLab/Bitbucket repositories onto the Platform



- Case 2: You want to copy files from your local environment into an existing project on the Platform
- Case 3: Your work is not in a version-controlled repository. Where do you start?

### 3.5.1 Moving Existing GitHub/GitLab/Bitbucket Repositories on the Platform

This is the easiest case. Make sure that under Settings, you have your Git provider credentials in place.

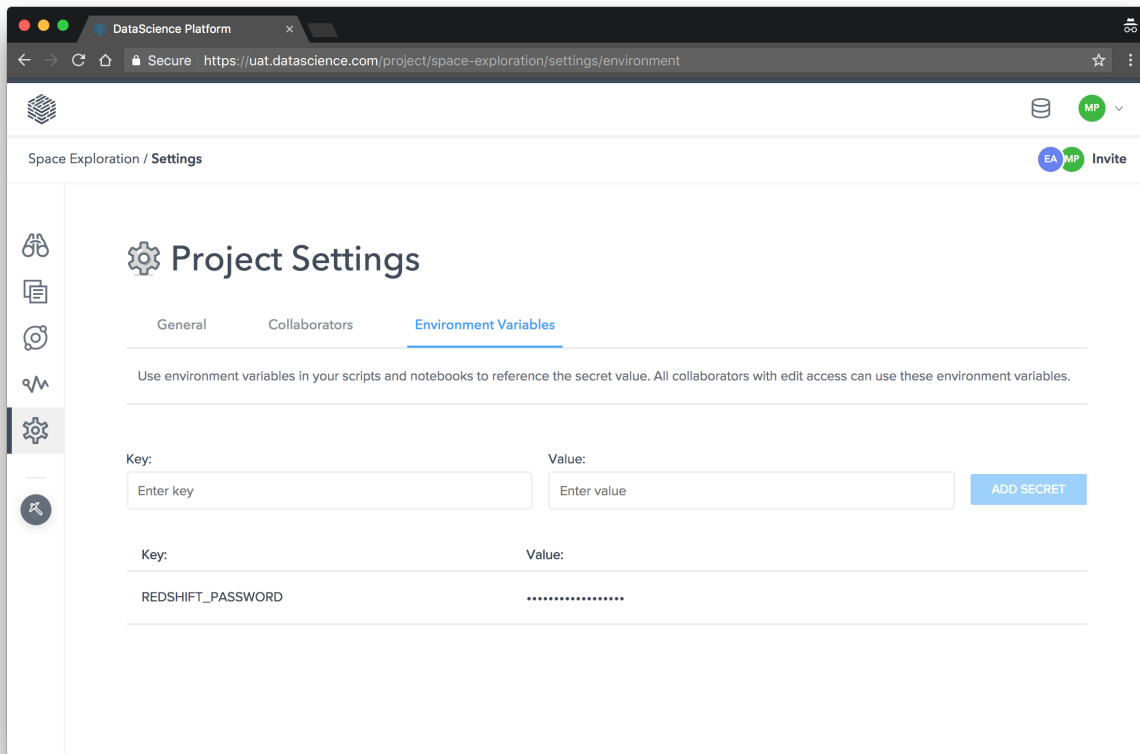
Once you have verified your credentials, go back to the Projects page, choose your Git provider and enter the name of the repo you want to migrate to the Platform.

### 3.5.2 Copying Files from Your Local Environment into a Project

If you have a written notebook or a script on your laptop and you want to move those files into an existing project, there are two methods you may follow:

- Method 1: Clone the repository of the project on your machine, `git add` the files, `git commit` them and push your branch to `remote`. Open a session on the Platform under that project and you should see that the new files are accessible in your project.
- Method 2: You can add files to your project by using the Upload button within your Jupyter session.

If you have multiple files that you want to move to an existing project, create a file archive (tar) and upload it to the Platform. From a Python Jupyter notebook, enter the following command to unpack the file:



```
!tar -xvf filename.tar
```

If you have compressed the file with `gzip`, you can unpack and decompress the file with a single command:

```
!tar -xjvf filename.tar.gz
```

### 3.5.3 Migrating Work That is Not in a Version-Controlled Repository

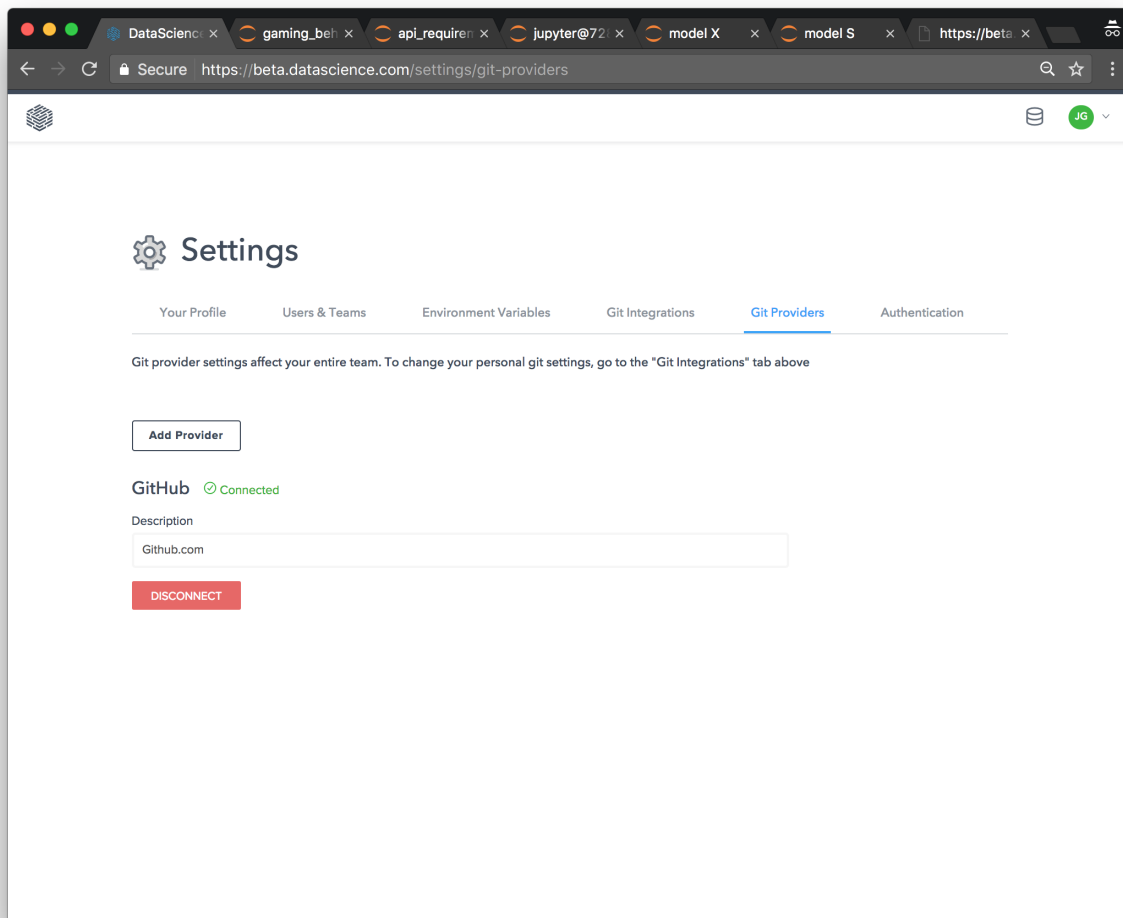
In this case, you have files in a folder either locally or in a remote environment that is not version-controlled.

### 3.5.4 Warnings

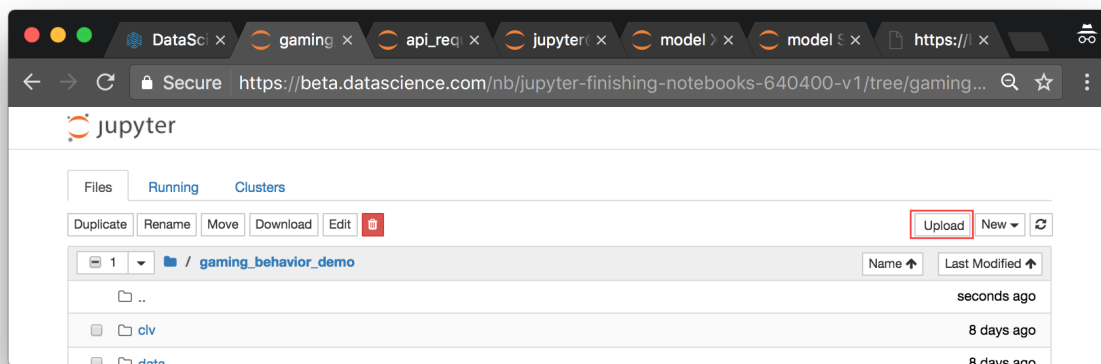
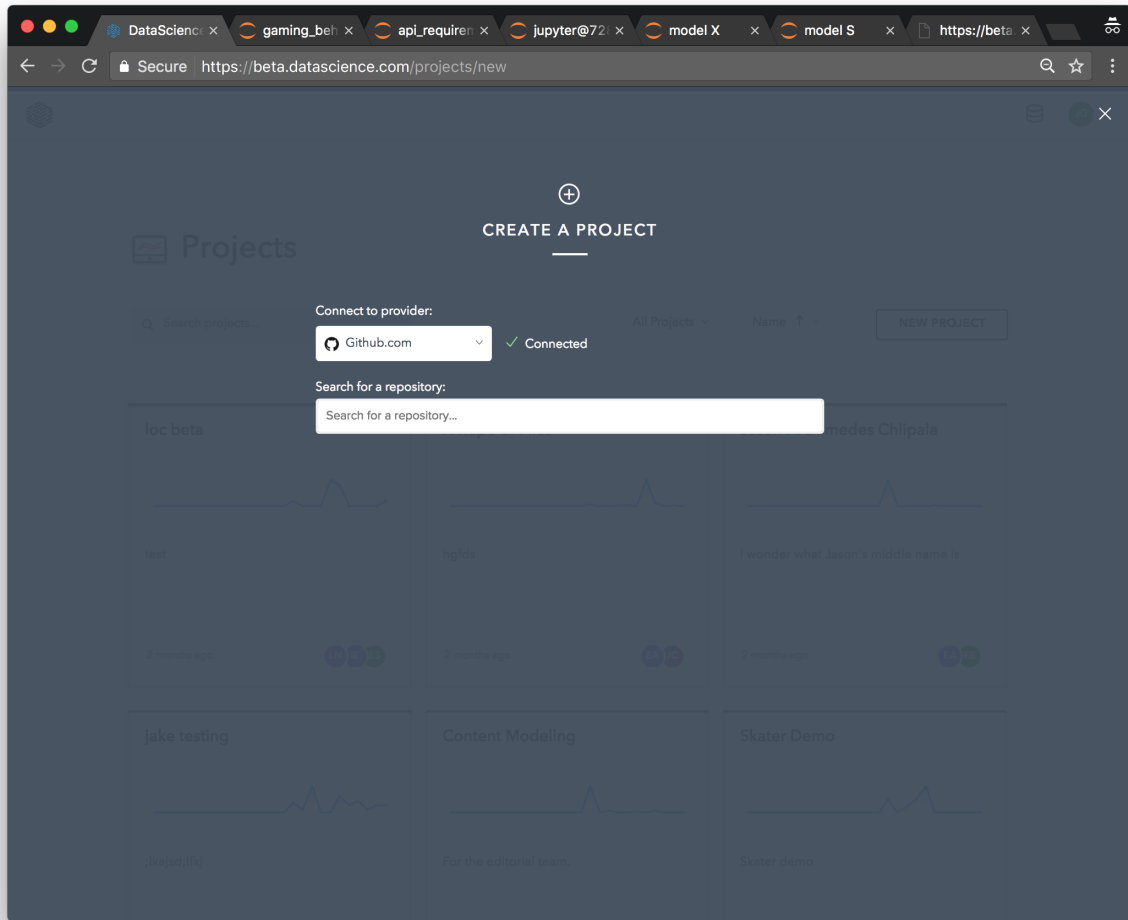
- Avoid copying or moving large data files in your project. If your team is using the cloud, put these files on a shared file system such as [Amazon AWS S3](#) or [Microsoft Azure Blob](#). The Docker containers are of finite size and you don't want to version control large data files. Github, for example, has a [file size limit of 100MB](#). Keep your repository under 1GB in size.

### 3.5.5 References

- [An introduction to the Git command line interface \(CLI\)](#)
- [An introduction to the Bitbucket command line interface \(CLI\)](#)









Every analysis on the DataScience.com Platform runs in an isolated container, much in the same way your laptop is isolated from your teammates' laptops. But, the key difference on the Platform is that containers are reproducible, sharable, and can be tailored to many different workflows.

The configurable elements of services are: environments and dependencies (the software you access in your analysis), compute resources (the RAM and CPUs available to the analysis), and environment variables (encrypted passwords and other configuration information available at runtime).

## 4.1 Environments and Dependencies

### 4.1.1 Introduction

Environments are customized, pre-installed collections of dependencies and packages that can be created by admins and distributed to users on the DataScience.com Platform.

### 4.1.2 Browsing environments

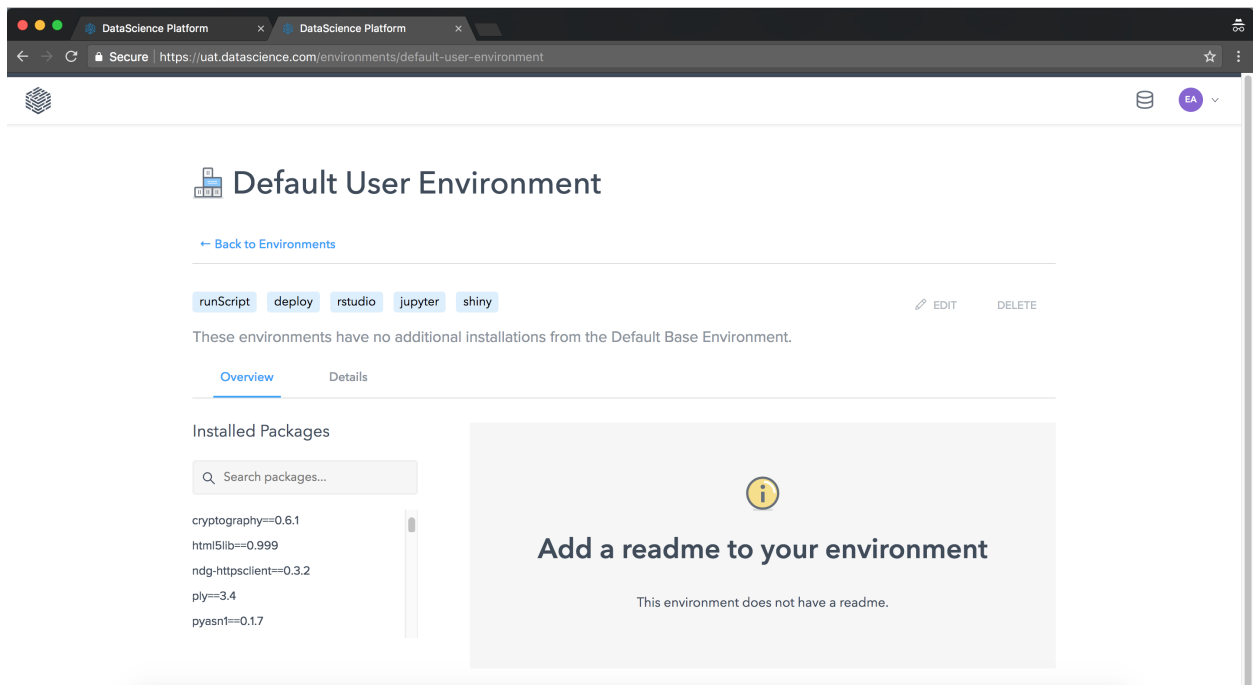
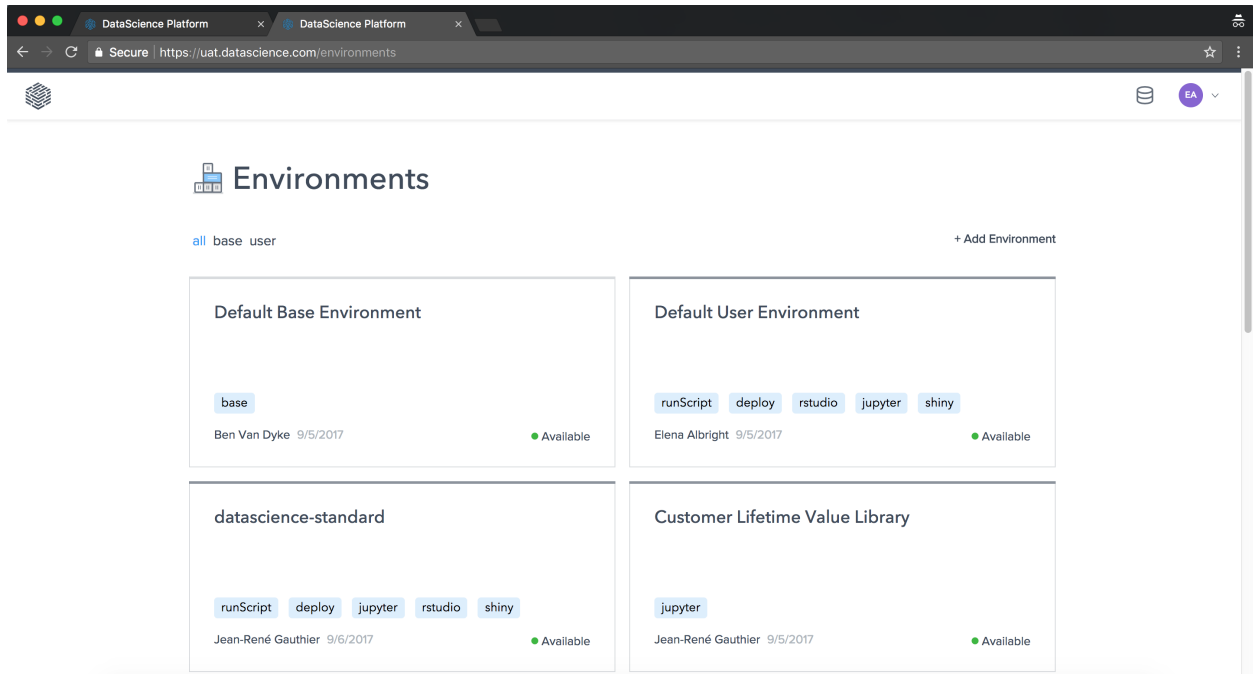
#### 4.1.2.1 List page

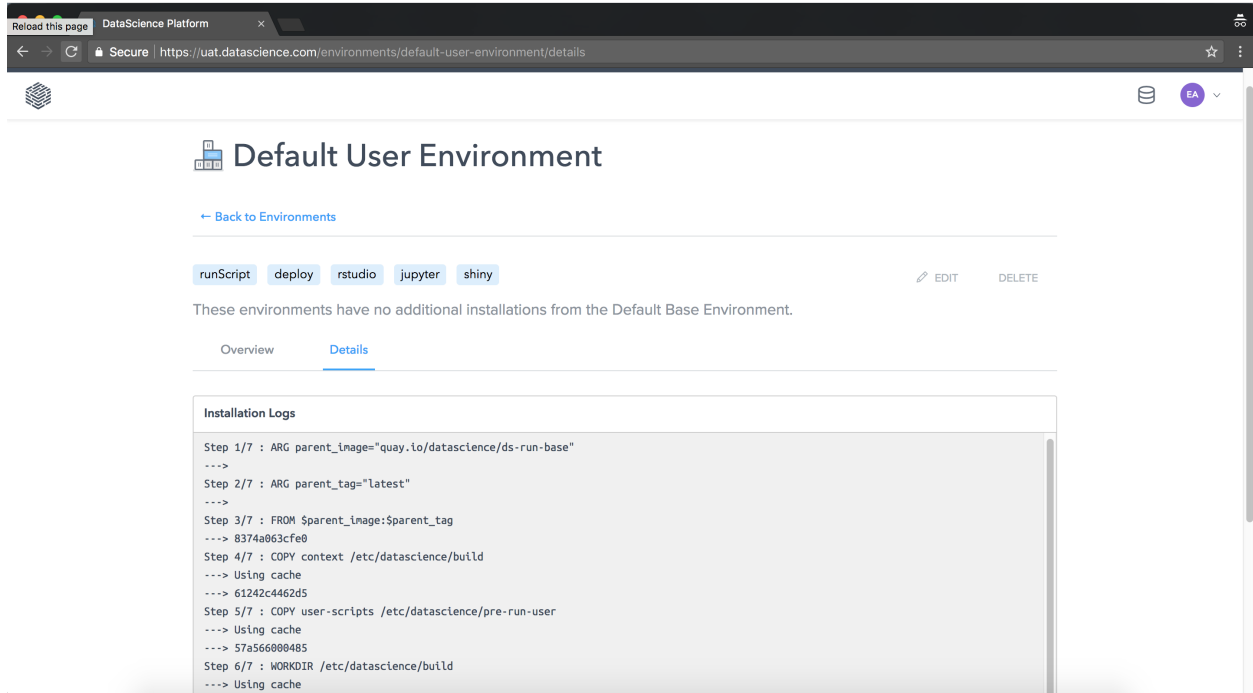
To learn more about what environments are available in your instance, navigate to the Environments page under your Avatar dropdown. On this page, you will see all of the environments in your instance and what tools are available for each of them. For more details, you can click on an environment card.

#### 4.1.2.2 Details page

For each environment, you can see a description and README provided by your Platform admin on the Overview tab. Search the installed package list to see the dependencies that will be pre-installed.

On the Details tab, you can see the installation logs and Dockerfile information.





### 4.1.3 Launching environments

To run analyses or create outputs on the Platform, you can launch Docker containers to host your work. When spawning containers, you can configure the environment that you want to run. Choose an environment from the dropdown menu. Only environments that are available for your tool will be available to select and run.

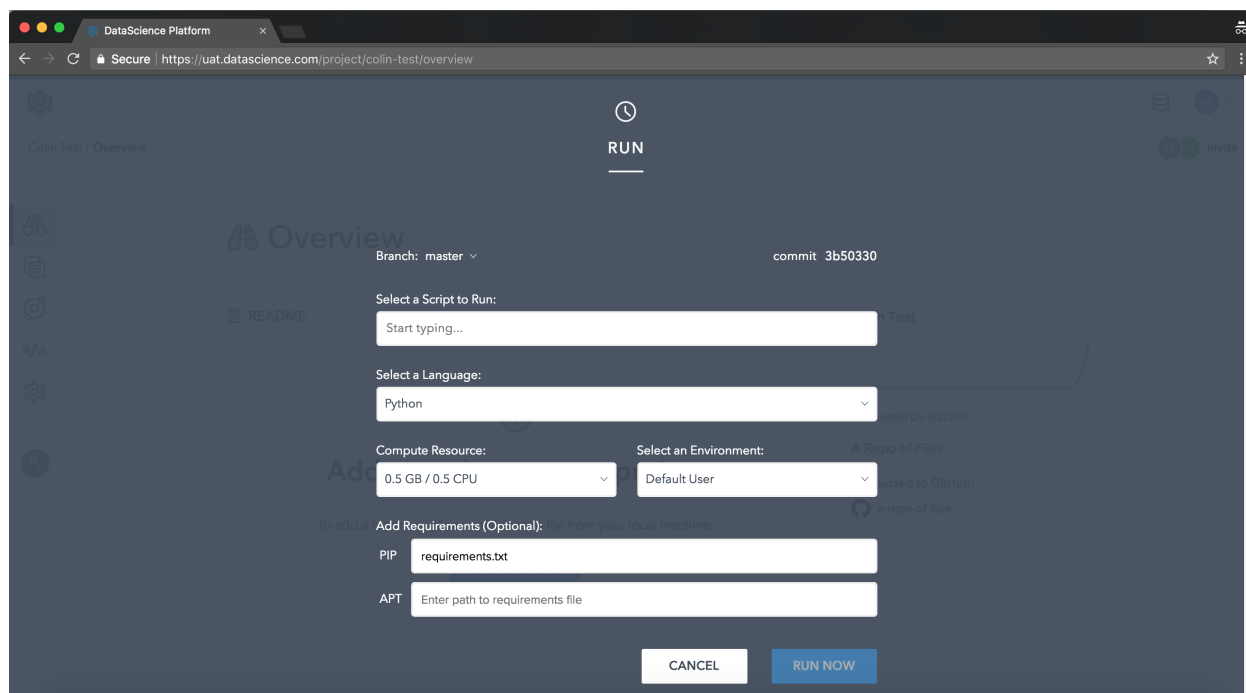
### 4.1.4 Adding additional requirements

When configuring your container, you can specify additional requirements to install at runtime by clicking the Add Requirements button on the action modals. Depending on the language selected, you'll find forms for `pip`, `R` (which runs `install.packages(...)`), and `apt` dependencies. When you include a list (in text file format) of packages for these installers, the platform will install them before running your code.

Notice that the form above points to a text file called `requirements.txt`. While you can call that file anything you want, it must be formatted as a different package name on each line. The `apt-get` and `R` installers accept only package names and will install the latest stable version. The `pip` installer accepts either package name or a version-locked name, like the example below:

```
# install locked version of plotly
plotly==2.0.12
# install latest version of seaborn
seaborn
```

For `pip` only, the comments in the example above are valid syntax.



## 4.2 Environment Management

### 4.2.1 Introduction

Environments are customized, pre-installed collections of dependencies and packages that can be created by Admins and distributed to Users on the DataScience.com Platform. Users require environments to be built in order to do work on the Platform. Therefore, it is critical that an Admin create environments during the installation process. Additional environments can be built anytime thereafter.

In this guide, you will learn how to build environments on behalf of Users.

### 4.2.2 Background

The DataScience.com Platform uses Docker to containerize workloads on your instance. Docker containers allow Users to spin up isolated work environments that have all of the software needed for their analysis pre-installed. A Docker container is a running instance created from a Docker Image. Docker Images are immutable files that define the runtime of containers. When a user runs a script or launches a Jupyter session on the Platform, they are running a Docker Image that is stored in an internal Docker Registry. The Environments feature allows Admin users to create their own Docker Images and submit them to the Docker Registry by writing a Dockerfile within the Platform interface.

If you are unfamiliar with Docker and Dockerfiles, check out Docker's documentation for more details. Please also refer to our Dockerfile Basics and Best Practices documentation for several example Dockerfiles and *best practices*.

### 4.2.3 What is an Environment?

An environment is defined by a Dockerfile and a pre-run script; it is associated with metadata such as name, description, and a README. There are two categories of environments: Base and User.

### 4.2.3.1 Base Environments

A Base environment contains many of the fundamental packages that are necessary for a DataScience.com Platform container to run and connect to data in your Instance. There are three types of Base environments: Default, Custom, and Hadoop-enabled.

#### 4.2.3.1.1 Default Base Environment

The Default Base environment must be created first. This environment is provided by DataScience.com; it contains the software that ensures containers will spawn and function successfully on the Platform. You cannot modify this environment's Dockerfile, but you can extend it when you create custom or Hadoop Base environments.

**Warning:** The Default Base environment cannot be edited or deleted.

#### 4.2.3.1.2 Customized Base Environments

Base environments can be customized to include the common languages and dependencies that you want to be readily available across environments. Custom Base environments can be created from each other by extending with Dockerfile commands.

### 4.2.3.2 User Environments

A User environment is an image that is launch-able in projects on the DataScience.com Platform, as they contain the tooling that is needed for User actions (launch a session, run/schedule a script, publish an application, deploy an API). This is an additional place where the Dockerfile can be extended with additional customization.

## 4.2.4 How to Create Environments

### 4.2.4.1 Before You Begin

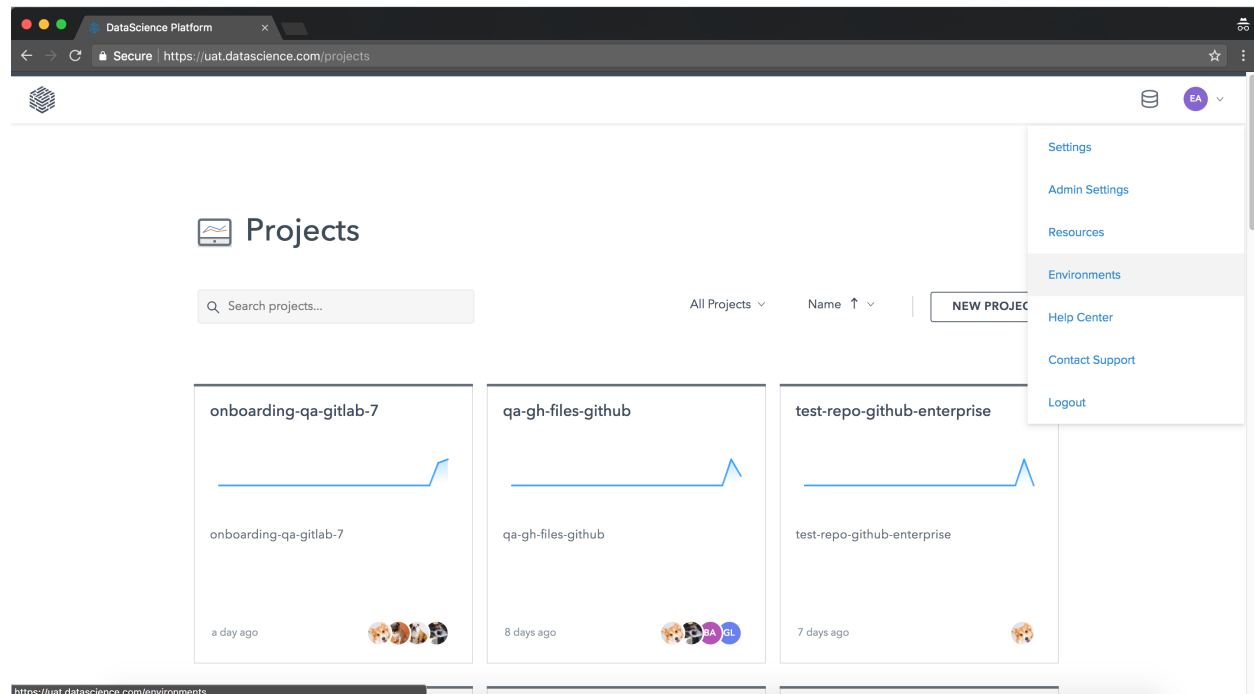
Prior to creating environments, please heed the following best practices to ensure a successful build:

- Do not build multiple environments (Base or User) concurrently.
- Please set up a Git repository to store Docker and context files used to build environments. This practice will not only make iteration and troubleshooting much easier, but also will enable tracking of an environment's history. Add the commit number and repo URL to the description for an environment.
- Environments should not be rebuilt frequently to ensure that Platform users have consistent, standardized workspaces. We recommend testing and updating versions of packages, languages, libraries, drivers, and tools once a year.
- To keep the frequency of updates low, we do not recommend that you rebuild environments to add small numbers of packages. The additional packages can be installed within an end user's session at runtime instead.
- If you want to replace an existing environment, build the new environment first and test it fully to ensure it performs as expected. Do this before deleting the older environment to be replaced.
- Before adding new packages to your environment, test the installation in a Jupyter, RStudio, or Zeppelin session first. This will allow you to identify any additional dependencies that may be needed for that package.

- If you are replacing the version of Python or R that comes with the Default Base environment (e.g. you want Python 3.6 or R 3.4.2), set up a Custom Base that has a minimal installation of the new version (few dependencies present) and then expand upon this Custom Base with the desired dependencies (e.g. comprehensive package list or Hadoop dependencies).

### 4.2.4.2 Default Base

After installation is complete, navigate to the Environments page via the Avatar dropdown in the top right corner.



When you arrive on this page for the first time, you will see a button that allows you to create the first environment: the Default Base Environment. When you click Add an Environment, the build process will be kicked off.

On the next page, you will be able to see the logs from the Docker build process. This should take a few minutes to complete. Once complete, click Confirm & Save. This will allow you to extend the environment in the future.

Once the Default environment is created, you can continue to customize your Base environment (recommended). At this stage, it is a good practice to create Base environments with the languages (and their versions), package managers, and common packages used by your organization.

### 4.2.4.3 Custom Base

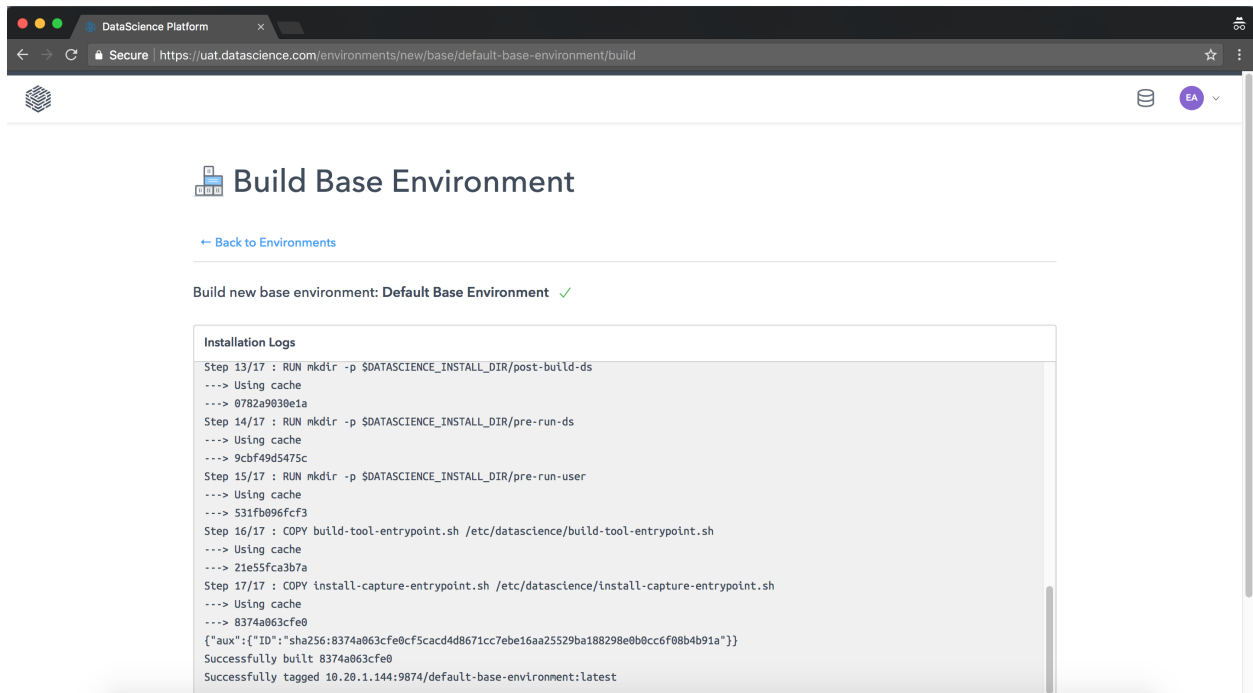
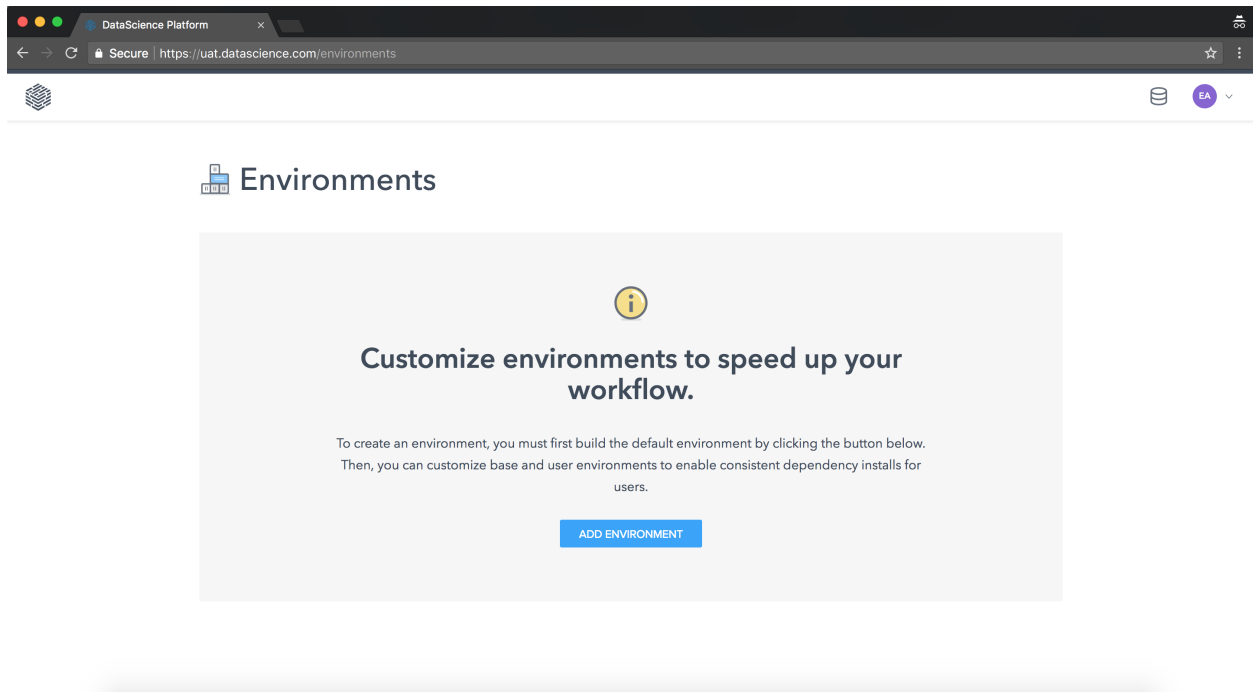
To create a custom Base environment, click Add Environment > Base Environment on the Environments screen.

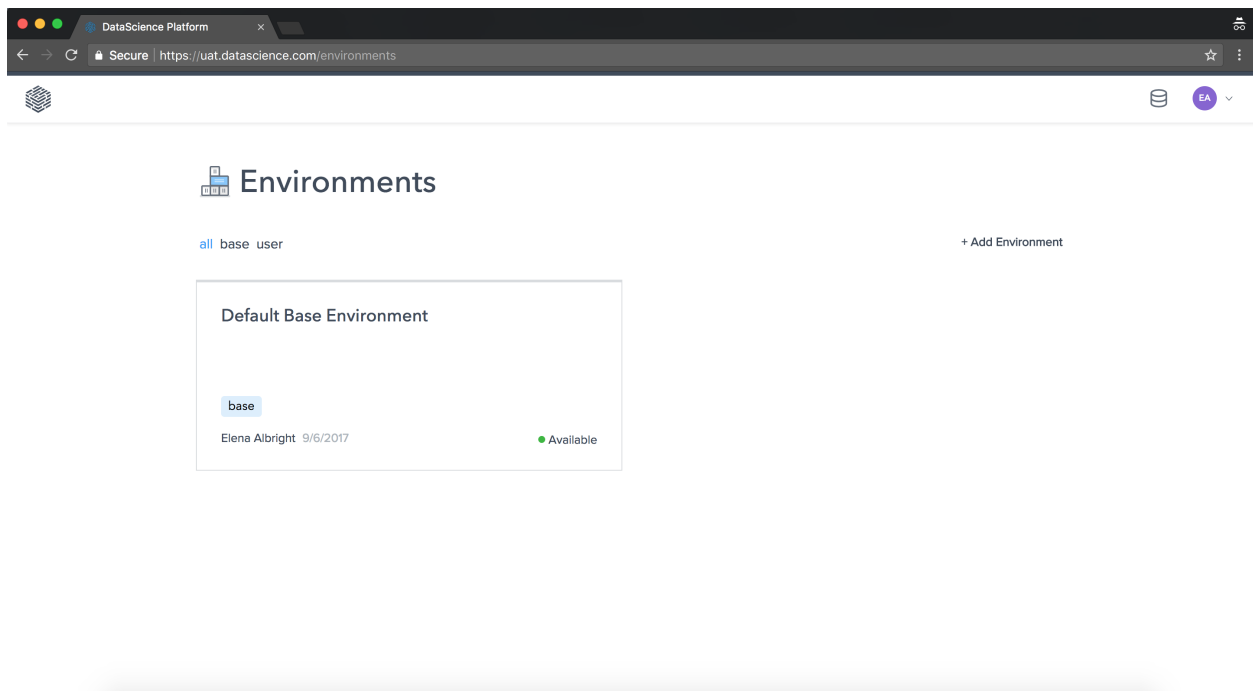
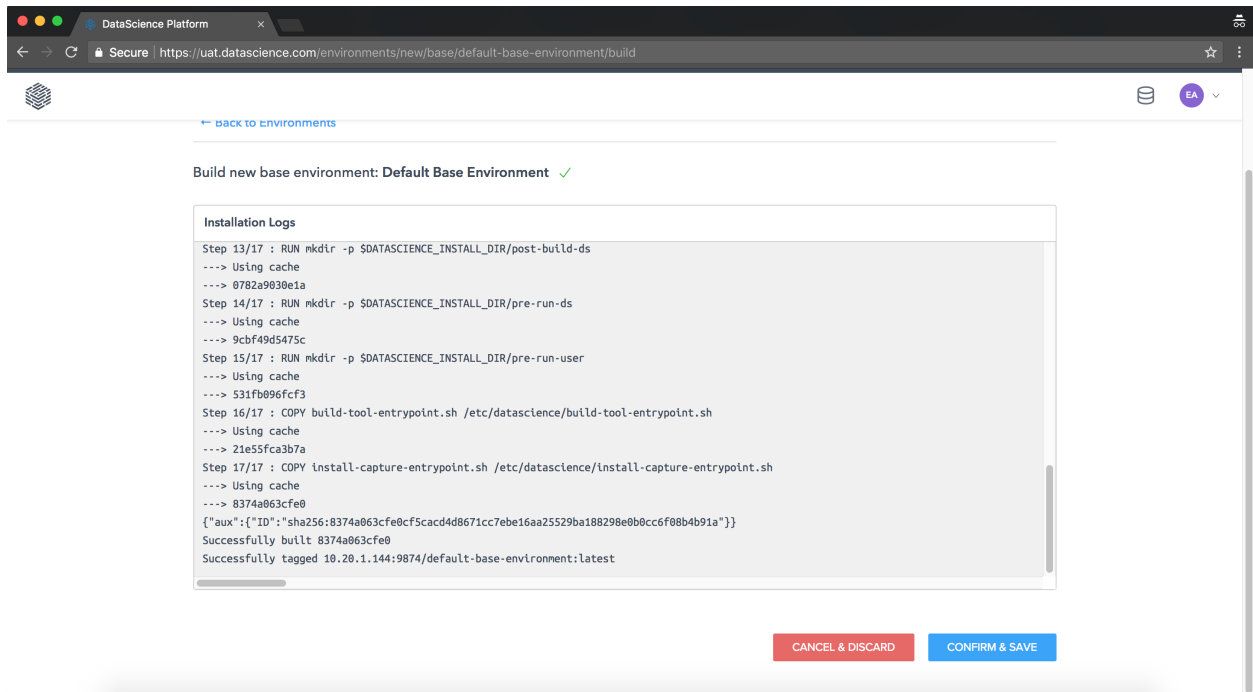
Give your new environment a name and description, and upload a README that tells your users a little more about how to use the environment and its intended purpose. Then, select Custom Dockerfile.

Choose another Base environment that you want to extend in the FROM selection. When you are finished with this environment, you will see it in this dropdown for future customization.

In the Dockerfile text area, enter Dockerfile commands for installation of your intended packages. Refer to our [Dockerfile Basics and Best Practices](#) documentation for more information about how to write Dockerfiles in the DataScience.com Platform. There are a few restrictions on the commands that you can enter in this area:







**New Base Environment**

[← Back to Environments](#)

Environment Name:  
Data Science Standard

Environment Description:  
This environment contains all of the dependencies and packages needed for all user environments.

README.md

☒ Custom Dockerfile

From:  
Select an environment...

Enter Dockerfile:

- No FROM
- No ENTRYPOINT
- No CMD
- No EXPOSE
- No absolute paths
- No tool installation

In the next step, you can upload a .tar file that contains all of the files that you reference in your Dockerfile context.

In the pre-run script text area, you can enter any command that you wish to execute at the start of runtime of a container.

When your new base environment is customized to your specifications, you are ready to move on to the build step. In this next page you can see the logs from the installation process. It's important to check these logs for any errors or skipped package installations. If the build does not complete to your satisfaction, click Cancel & Discard to start over.

When you are satisfied with your build, click Confirm & Save to make it available.

#### 4.2.4.4 Create a User Environment

To make tools available to the users of your instance, create a User environment by selecting Add Environment > User Environment on the Environments page.

As with the Base environment, enter a name, description, and README that will help your users understand what is included in this environment and its intended use.

As with custom Dockerfile Base environments, you can select a Base environment to extend and add additional commands in the Dockerfile text area. Use the Upload .tar button to submit context files referenced in your Dockerfile.

Next you must select the tool environments that you want to build to make available to users. In general, it's good to build as many of these as makes sense for the packages that you've installed. For example, if your environment

From:  
Default Base Environment

Enter Dockerfile: ⓘ

```

RUN pip install -r requirements.txt

# install the r packages from txt file
RUN echo ".First <- function() {options(repos = c(CRAN = 'https://cran.rstudio.com/'))}" >> /usr/lib/R/library/base/R/Rprofile && \
cat cran.txt | awk '{system("/usr/bin/Rscript ./install.r \"$1\")}' && \
R -q -e 'devtools::install_github(\"hadley/xm12\")'
```

Upload .TAR datascience-standard.tar

Enter Pre-Run Script:

NEXT STEP: BUILD →

## Build Base Environment

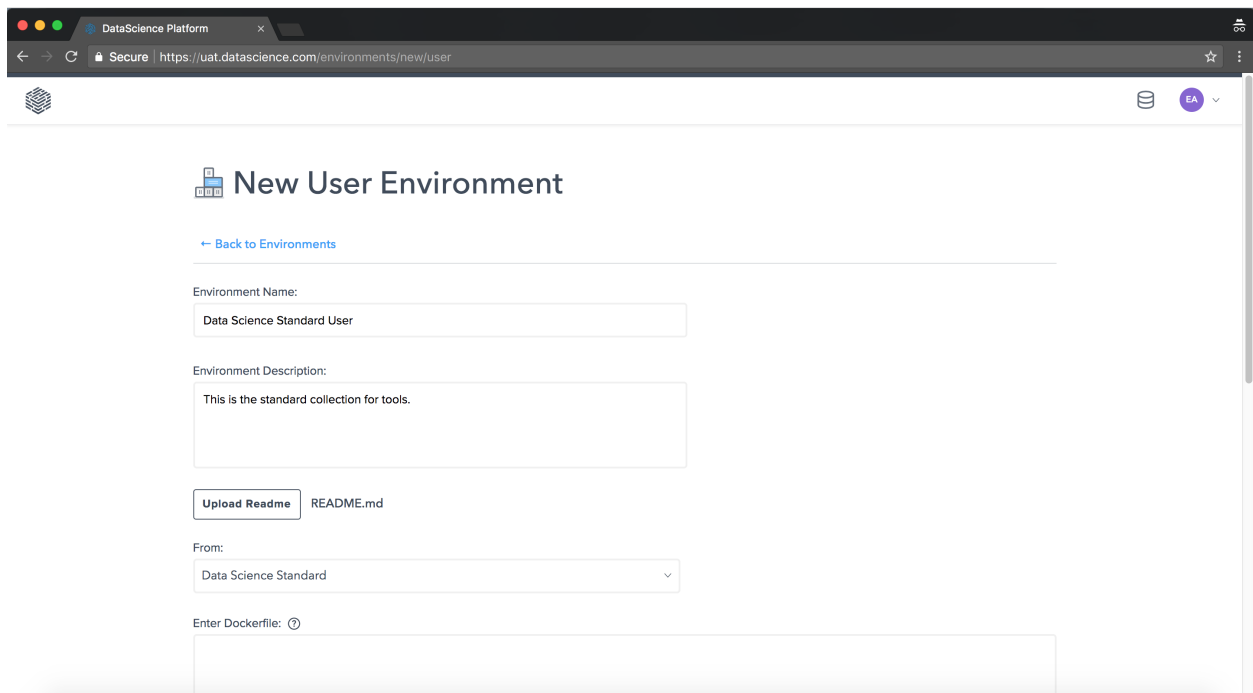
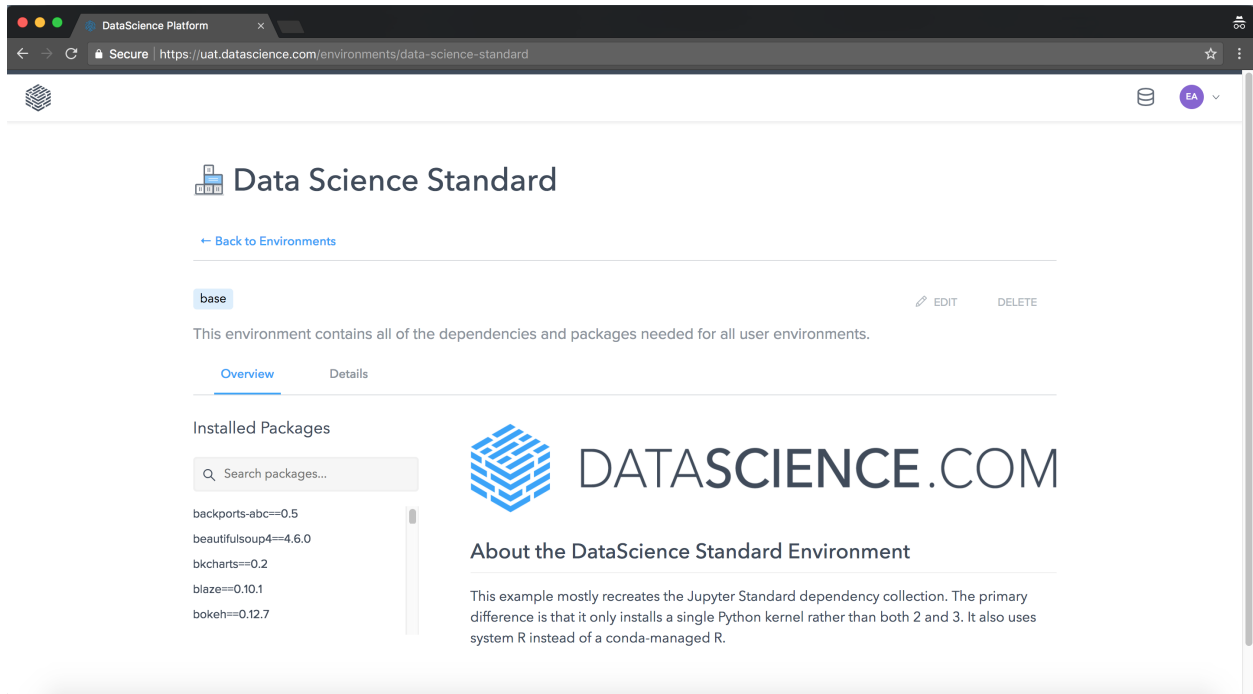
[← Back to Environments](#)

Build new base environment: Data Science Standard ●●●

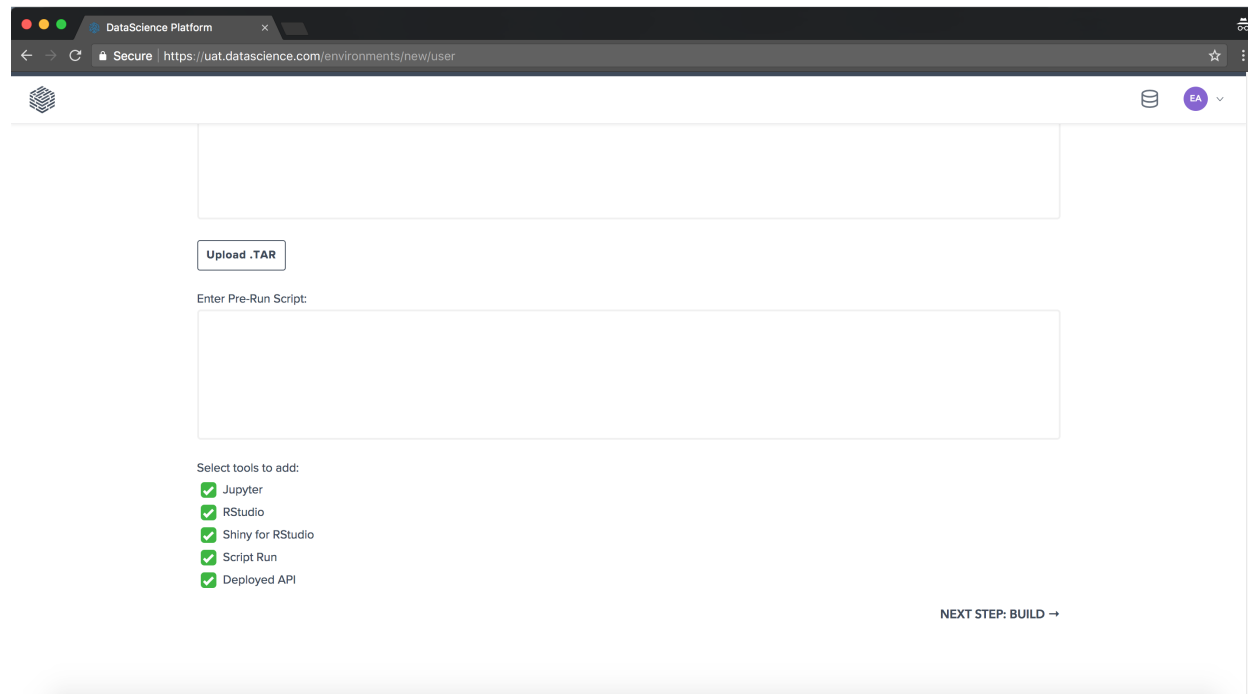
Installation Logs

```

Downloading certifi-2017.7.27.1-py2.py3-none-any.whl (349kB)
Collecting backports_abc==0.4 (from tornado==4.3->bokeh->-r requirements.txt (line 22))
Downloading backports_abc-0.5-py2.py3-none-any.whl
Collecting smmap2==2.0.0 (from gitdb2==2.0.0->gitpython->-r requirements.txt (line 24))
Downloading smmap2-2.0.3-py2.py3-none-any.whl
Collecting jupyter_core (from nbformat==4.2->plotly->-r requirements.txt (line 28))
Downloading jupyter_core-4.3.0-py2.py3-none-any.whl (76kB)
Collecting traitlets==4.1 (from nbformat==4.2->plotly->-r requirements.txt (line 28))
Downloading traitlets-4.3.2-py2.py3-none-any.whl (74kB)
Collecting jsonschema==2.5.0,>=2.4 (from nbformat==4.2->plotly->-r requirements.txt (line 28))
Downloading jsonschema-2.6.0-py2.py3-none-any.whl
Collecting ipython_genutils (from nbformat==4.2->plotly->-r requirements.txt (line 28))
Downloading ipython_genutils-0.2.0-py2.py3-none-any.whl
Installing collected packages: numpy, pytz, python-dateutil, pandas, numexpr, pyparsing, cycler, subprocess32, functools32, matplotlib, sc
Running setup.py install for subprocess32: started
Running setup.py install for subprocess32: finished with status 'done'
Running setup.py install for functools32: started
Running setup.py install for functools32: finished with status 'done'
```



contains Python 3 and various Python packages, it makes sense to build Jupyter, Deployed API, and Script Run. If your environment has a mixture of both Python and R packages, it makes sense to build all available tools.



The screenshot shows a web browser window with the address bar displaying 'https://uat.data science.com/environments/new/user'. The page has a header with the DataScience Platform logo and a user profile icon labeled 'EA'. The main content area includes a large empty text box at the top, followed by an 'Upload .TAR' button. Below this is a section labeled 'Enter Pre-Run Script:' with another large empty text box. Underneath is a list of tools to add, each with a green checkmark: Jupyter, RStudio, Shiny for RStudio, Script Run, and Deployed API. At the bottom right of the form is a button labeled 'NEXT STEP: BUILD →'.

Once you are satisfied with your environment’s specifications, click Next Step: Build to move on to the build step where you can observe the installation logs and the tool building progress.

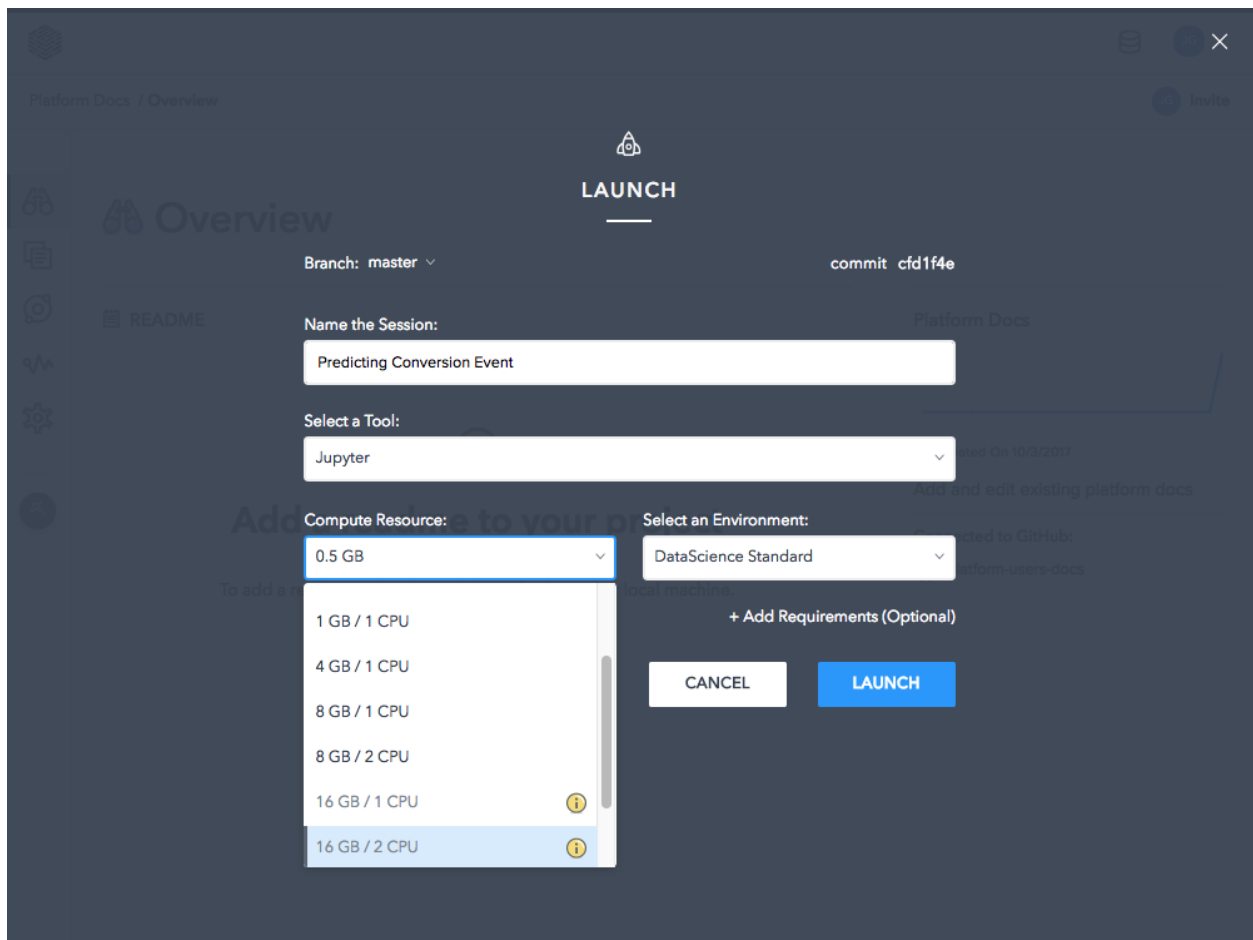
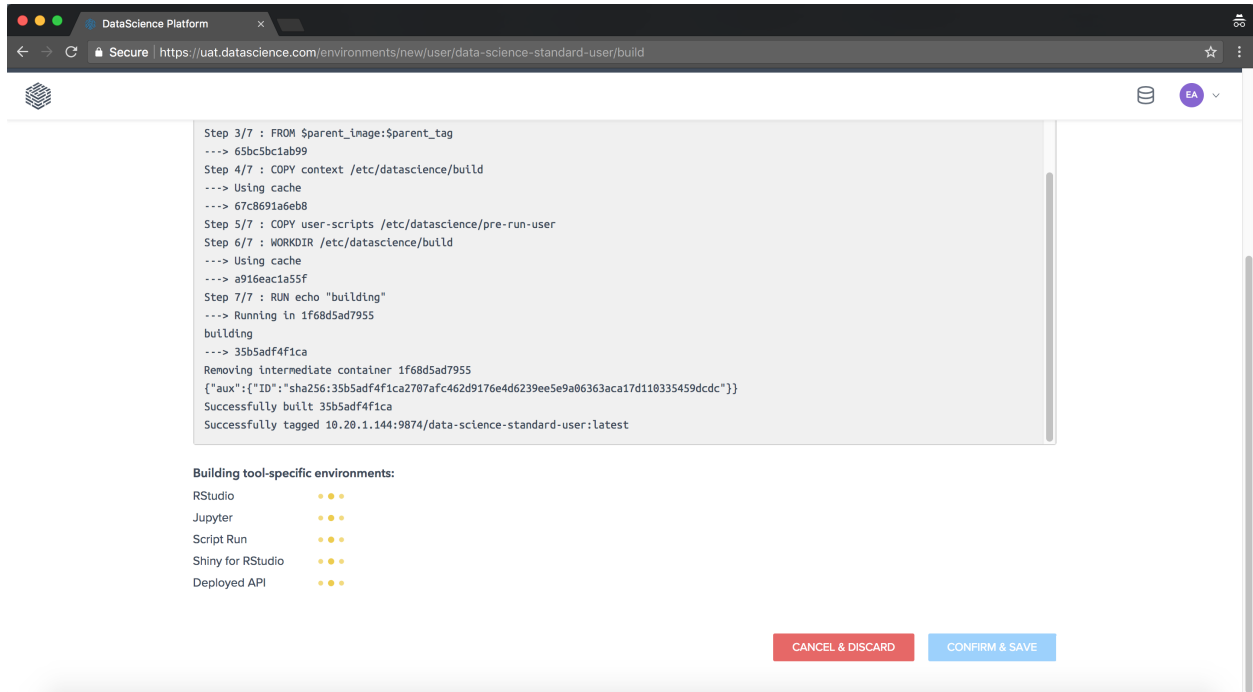
Click Cancel & Discard to start over; click Confirm & Save to make the environment available to Users. Once this is complete, Users will be able to launch these tools within their projects. See the user documentation on [Environments and Dependencies](#) for more information.

## 4.3 Compute Resources

Before any analysis, you’ll have the option to select your compute resource. This option specifies the RAM and CPUs available to your analysis. For example, if you chose 16 GB/4 CPU from the Compute Resource dropdown, you’ll have 16 GB of RAM and 4 CPUs available to your analysis.

These standard resource sizes represent available space on the servers that your admin provisioned to run the Platform. These long-lived servers are often referred to as the “standing pool.” Since these servers are always on and ready to accept a new analysis, you can expect your analysis to launch in a matter of seconds.

When clusters start to fill up with requests for resources, some of the compute resource options will become unavailable. In the snapshot below, you see an example where the standing pool servers are utilized to the point where the 16 GB/1 CPU and 16 GB/2 CPU configurations are no longer available. In such a scenario, you could request your administrator to increase the standing pool resources by adding nodes to the cluster, wait for services to end and resources to free up, or use on-demand compute resources (if this feature is enabled for your installation). Such on-demand compute resources are the topic of the next section.

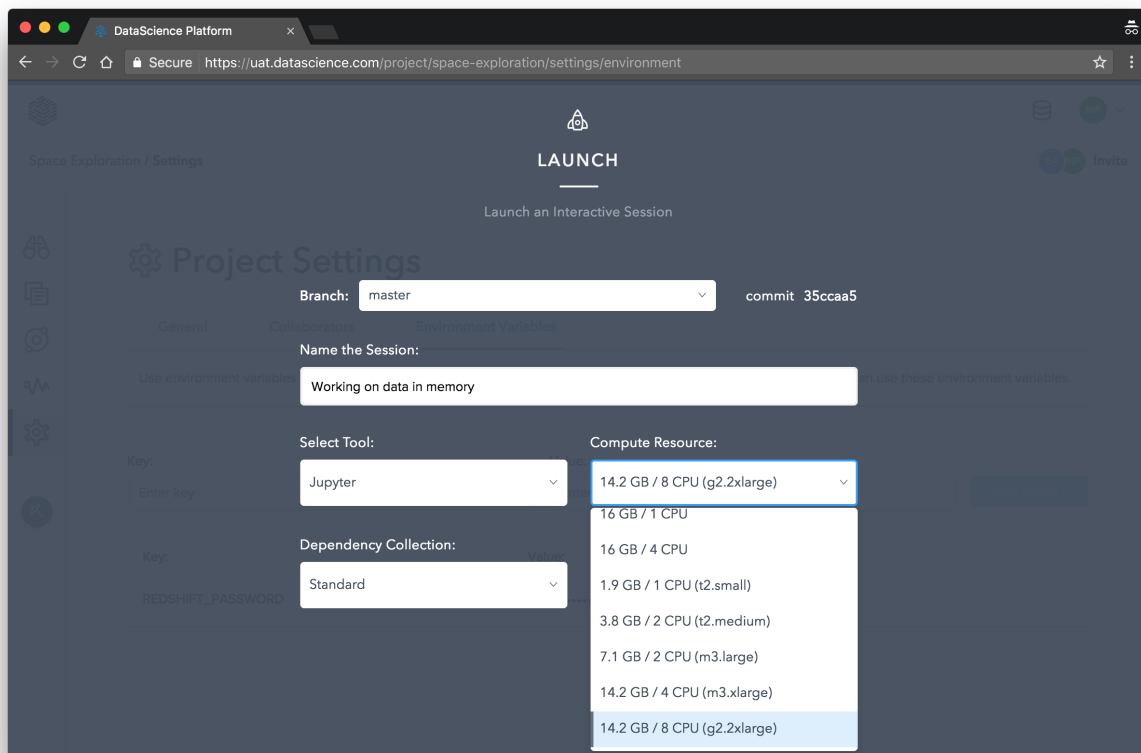


### 4.3.1 On-demand Compute Resources for VPC Installations

**Warning:** The following feature is applicable only to Amazon Web Services (AWS) installations.

Installations on Amazon Web Services (AWS) may take advantage of on-demand compute. In the image below, there are a number of AWS EC2 instance sizes available. When you choose one of the on-demand options, the Platform will create a new EC2 instance and run your analysis on it. When your analysis is finished, the Platform will destroy the EC2 instance and you'll only be billed (by AWS) for the time you used it.

Creating brand new EC2 instances can take several minutes to start and you will see a run status of `Provisioning: Creating` while the server is being created.



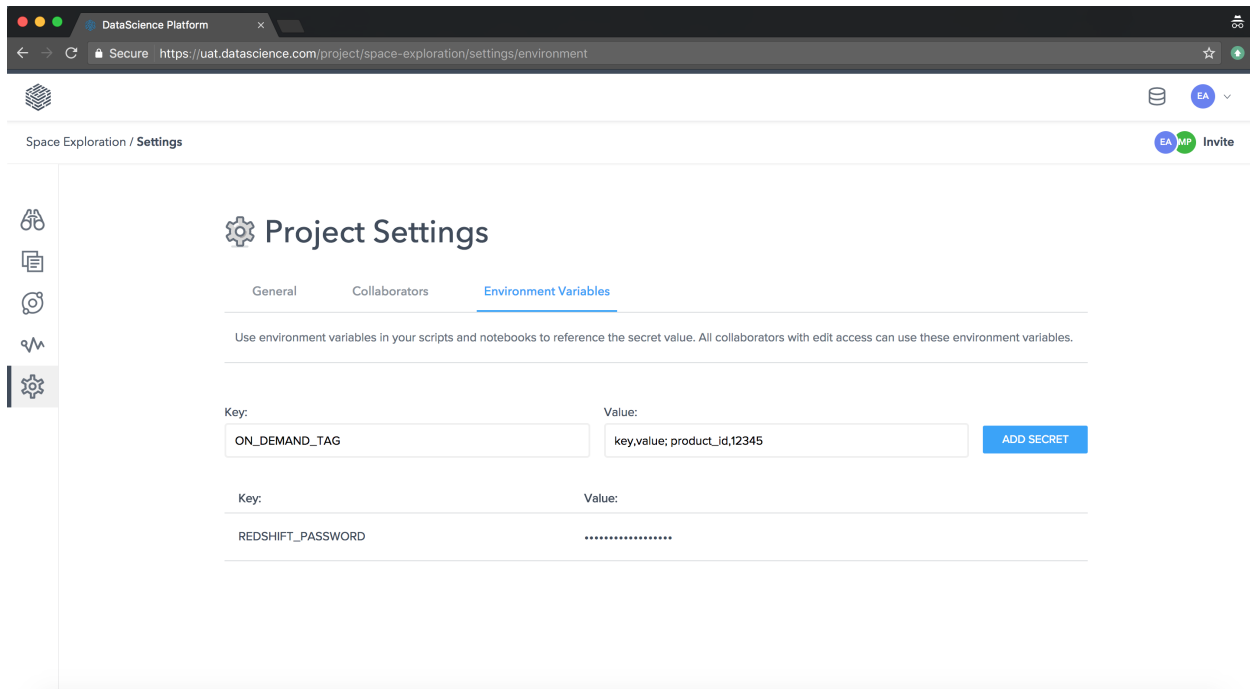
### 4.3.2 Tag Management for On-Demand Resources

AWS allows metadata to be assigned to EC2 instances via tags. Tags are useful for resource categorization, tracking, and management. See more information in the [Amazon documentation](#).

The DataScience.com Platform allows users to pass in custom tags via project-level *environment variables*. To add a custom tag, navigate to Project Settings > Environment Variables. Enter a variable with the key `ON_DEMAND_TAG`. The value of this key should be a semi-colon-separated list of key-value pairs that are separated by a comma.

As an on-demand instance is provisioned, the value tied to the project variable with the key `ON_DEMAND_TAG` will be passed into the EC2 metadata tag.





**Note:** Amazon has a number of restrictions for tags. Consult [their documentation](#) for more information.

#### **Warning: Duplicate tags cause provisioning error**

Tags with duplicate keys will be de-duplicated based on the first key that is received.

Be sure that each key in your environment variable's value appears only once. Additionally, ensure that your admin has not added other fixed tags with any of the same keys as your custom tags.

## 4.4 Using Spark on the Platform

The DataScience.com Platform includes an Apache Spark integration. Users can connect to a pre-configured remote Spark cluster via Jupyter and RStudio.

### 4.4.1 Jupyter

Spark integration for Jupyter is supported via the following Sparkmagic kernels:

- Spark (Scala)
- PySpark (Python 2)
- PySpark3 (Python 3)
- SparkR (R)

DataScience.com Platform Jupyter notebooks connect to Spark using Livy, an open source REST interface for interacting with Apache Spark. Livy runs in an embedded mode in our integration and connects to Spark via cluster mode.

Connecting to Spark via cluster mode means that each Spark session driver will run on the remote Spark cluster.

## 4.4.2 RStudio

RStudio support uses the **sparklyr** or the **sparkr** library.

---

**Note:** For workflows that expect low latency (e.g. deployed APIs or Shiny applications), we do not recommend querying directly against your data source with Spark, as the query response times can be significant. Instead, you could use Spark within an Interactive Session to write pre-queried data to another location for use in a deployed API or Shiny application.

---

## 4.4.3 Spark Usage

### 4.4.3.1 Sparkmagic

Sparkmagic has a great introduction on usage [here](#).

A full list of Spark configurations for Spark on YARN can be found [here](#). These are configurable via the `%%configure sparkmagic` JSON object's `conf` key/value property.

### 4.4.3.2 Spark in RStudio

You can find documentation on how to access Spark via the Sparklyr library from RStudio [here](#).

---

**Note:** We recommend users launch at least a 2GB/1 CPU compute resource for Spark sessions.

---

## 4.5 Best Practices: Choosing the Right Container Size

In this article, we discuss how to allocate resources for Docker containers running on the DataScience.com Platform. If you're new to Docker containers, please see [Docker's documentation](#) for a helpful overview. For the purposes of this article, all you need to know is container resource allocation.

All DataScience.com Platform services (launching a session, running a script, scheduling a job, deploying a model, etc.) run on containers. Containers are sandboxed environments running on your on-premise infrastructure or in the cloud. When you launch a service on the Platform, a new container is created on one of your host machines. You can specify the maximum amount of memory and CPU that the service is allowed to use on the host.

Though services running on the same host are isolated from each other, they all share the host's CPU and memory. If too many resource-intensive containers are competing on the same host, the following may occur:

- Services may slow down or error out.
- You may be unable to launch new services on the host.

### 4.5.1 Strategic Resource Allocation

To avoid complications, it's important to allocate resources for your containers strategically. You want to ensure that your container runs with enough resources to run your code and you want to write code that requires as few resources as possible.

To determine the CPU and memory requirements of your code, use the following profiling tools:

- `memory_profiler` for memory usage in Python
- `psutil` for CPU usage in Python
- `profr`, `profviz` and RStudio profiler for memory usage in R
- `top` or `htop` for memory and CPU usage or any script

To determine the amount of resources available in your environment, Admin users can view the DataScience.com Platform Resources page, accessible through the avatar dropdown.

To avoid overutilizing resources:

- Schedule jobs at off-peak times.
- Terminate and close unused models and sessions.
- (If enabled) use on-demand resources when few resources are available.

### 4.5.2 Code Best Practices

To reduce the footprint of your code, here are some tips:

- Use out-of-core tools when working with large datasets, such as `dask` for Python or `BigMemory` for R.
- Use sparse matrices when working with sparse data.
- Use batch training when possible.
- Delete used objects and object references to enable garbage collection.
- Employ vectorization techniques whenever possible (available through libraries like `pandas` and `NumPy`).
- Make use of libraries that implement efficient data structures and algorithms and avoid manually implementing machine learning algorithms unless necessary.
- Omit useless variables from learning algorithms and models. Use feature selection algorithms to successfully subset your feature space.
- Generally speaking, use Python 3 instead of Python 2. Python 3 contains several optimizations over Python 2.
- Use generators when possible to avoid holding iterables in memory.
- If reusing a model, load serialized objects in lieu of retraining, or initialize models with pretrained weights.
- Cache intermediary results.
- Only import what you need. For example: in Python, if you're just using a single function from a large library, just import the function.
- Push numerical calculations to more efficient languages like C/C++ or FORTRAN (see `Cython`, `Rcpp`).

## 4.6 Best Practices: Using Dependency Files

### 4.6.1 What are Dependency Files?

If you are installing packages during a session (either via `pip` or `install.packages`), read this article on creating and using dependency files.

In a nutshell, dependency files contain a list of the libraries and packages installed in a project environment. It's good practice to keep dependency files for each project you have, regardless of whether you are using our pre-built dependency collections.

Dependency files are very handy when you want to re-create the environment in which you developed your model, whether on the Platform or on your laptop. Dependency files are necessary to create the environment of your deployed API or scheduled run. This is especially true if you install extra packages during a session.

The Platform currently supports dependency files for (i) `pip` (Python), (ii) `R`, and (iii) `apt`.

### 4.6.2 How to Create Dependency Files in a Jupyter Session

#### 4.6.2.1 Creating a pip Dependency File in a Jupyter Python Session

The easiest way to create a complete dependency file in a Python project is to use the `!pip freeze` command in a Jupyter notebook session. As you work in your Jupyter session, you will likely install packages. Run the `pip freeze` command when you are ready to either close your session or deploy your model (don't forget to Sync!). We show the command in the snapshot below.

In the image, you can see that all packages have a `==` sign. This denotes the specific version of the package installed in your environment. When you use a `pip` requirements file to install these libraries in a new environment, you can always relax the constraint `==` by using the `>=`, `>`, `<`, and `<=` signs. Below is an example with the Python library `pandas`:

```
pandas==0.15.2 # exact version match.
pandas>=0.15.2 # any versions of pandas greater or equal to 0.15.2
pandas # install the most recent version of 'pandas' available on `pypi` <pypi.python.
↪org/pypi>`__
```

You can find more on the topic of `pip` requirements file format in [pip documentation](#).

#### 4.6.2.2 Creating a Dependency File in an R Jupyter Session

In `R` sessions, you can get a list of the installed packages by calling `installed.packages()` in a notebook cell. The snapshot below displays how you can do this within a Jupyter session. Note that for `R`, the Platform installer will only accept the package names in the dependency file and will install the latest stable version.

Make sure you (i) list one package per line and (ii) do not include the version number.

#### 4.6.2.3 Apt Dependency Files

In addition to the `pip` and `R` package managers, you can also create a dependency file for `apt`. `Apt` stands for Advanced Package Tool and is a set of tools for managing Debian packages. (Note that the Platform runs the [Debian OS](#)). If you want to install `apt` dependencies, we recommend listing these dependencies in a file called `requirements_apt.txt`. You can do so directly in the Platform by opening a new text file in a Jupyter session. For `R`, the `apt` installer will install the latest stable version of each package listed in `requirements_apt.txt` file.

```
In [1]: pip freeze
asn1crypto==0.22.0
backports-abc==0.5
backports.shutil-get-terminal-size==1.0.0
backports.ssl-match-hostname==3.5.0.1
beautifulsoup4==4.5.3
blaze==0.10.1
bokeh==0.12.5
boto==2.47.0
boto3==1.4.4
botocore==1.5.55
certifi==2017.4.17
cffi==1.10.0
chardet==3.0.2
click==6.7
cloudpickle==0.3.1
contextlib2==0.5.5
cryptography==1.8.1
cycler==0.10.0
Cython==0.25.2
cytoolz==0.8.2

In [2]: pip freeze > requirements.txt

In [ ]:
```

**Jupyter model X** Last Checkpoint: 2 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help Trusted R

In [29]: `# get a list of installed packages`  
`packages <- installed.packages()`

In [30]: `packages[1:5,]`

Package	LibPath	Version	Priority	Depends	Imports	LinkingTo	Suggests	Enhances	License	License_is_FOSS
BH	/opt/conda/envs/python3/lib/R/library	1.62.0-1	NA	NA	NA	NA	NA	NA	BSL-1.0	N
DBI	/opt/conda/envs/python3/lib/R/library	0.6-1	NA	R (>= 2.15.0), methods	NA	NA	covr, hms, knitr, magrittr, rprojroot, rmarkdown, RSQLite (>= 1.1-2), testthat, xml2	NA	LGPL (>= 2)	N
IRdisplay	/opt/conda/envs/python3/lib/R/library	0.4.4	NA	R (>= 3.0.1)	repr	NA	testthat, withr	NA	MIT + file LICENSE	N
IRkernel	/opt/conda/envs/python3/lib/R/library	0.7.1	NA	R (>= 3.2.0)	repr (>= 0.4.99), methods, evaluate (>= 0.5.4), IRdisplay (>= 0.3.0.9999), pbdZMQ (>= 0.2-1), crayon, jsonlite (>= 0.9.6), uuid, digest	NA	testthat, roxygen2	NA	MIT + file LICENSE	N
KernSmooth	/opt/conda/envs/python3/lib/R/library	2.23-15	NA	R (>= 2.5.0), stats	NA	MASS	NA	Unlimited		N

In [36]: `# Remove the double quotes from the package name and write the Package column to a csv file.`  
`write.table(packages[,c('Package')], file='requirements_r.txt', quote=FALSE, row.names=FALSE, col.names=FALSE)`

The format of the `requirements_apt.txt` file is the same as for the R package manager: (i) list one package per line and (ii) do not include the package version.

Here's an example of the content of a short `requirements_apt.txt` file:

```
r-base libreadline-dev gfortran
```

### 4.6.3 Using Dependency Files on the Platform

In the previous section you learned how to create dependency files. Now you will learn why you should use these dependency files and how you can use them in your workflow.

#### 4.6.3.1 In a Jupyter or RStudio Session

Dependency files are particularly useful when you are migrating work on the Platform. Let's suppose you have developed a model on your laptop and you want to move it onto the Platform. Reproducing your laptop Python environment on the Platform is easy if you captured the dependencies via `pip freeze`. Just run the following command on the Platform in a notebook cell:

```
!pip install -r requirements_python.txt
```

The packages on the Platform will match the ones you have used in your local/dev environment.

Dependency files are very useful when creating (or re-creating) an environment. In an R Session, you can also install many packages from a `requirements_r.txt` file. In a Jupyter notebook cell, run the following command where the file `requirements_r.txt` was created previously:

```
packageList <- read.csv('requirements_r.txt', header=FALSE, col.names=c('packages'))
packageList <- as.vector(packageList[,])
lapply(packageList, install.packages(packageList), character.only=T)
```

The same three commands can be executed within an RStudio session.

#### 4.6.3.2 When Deploying an API

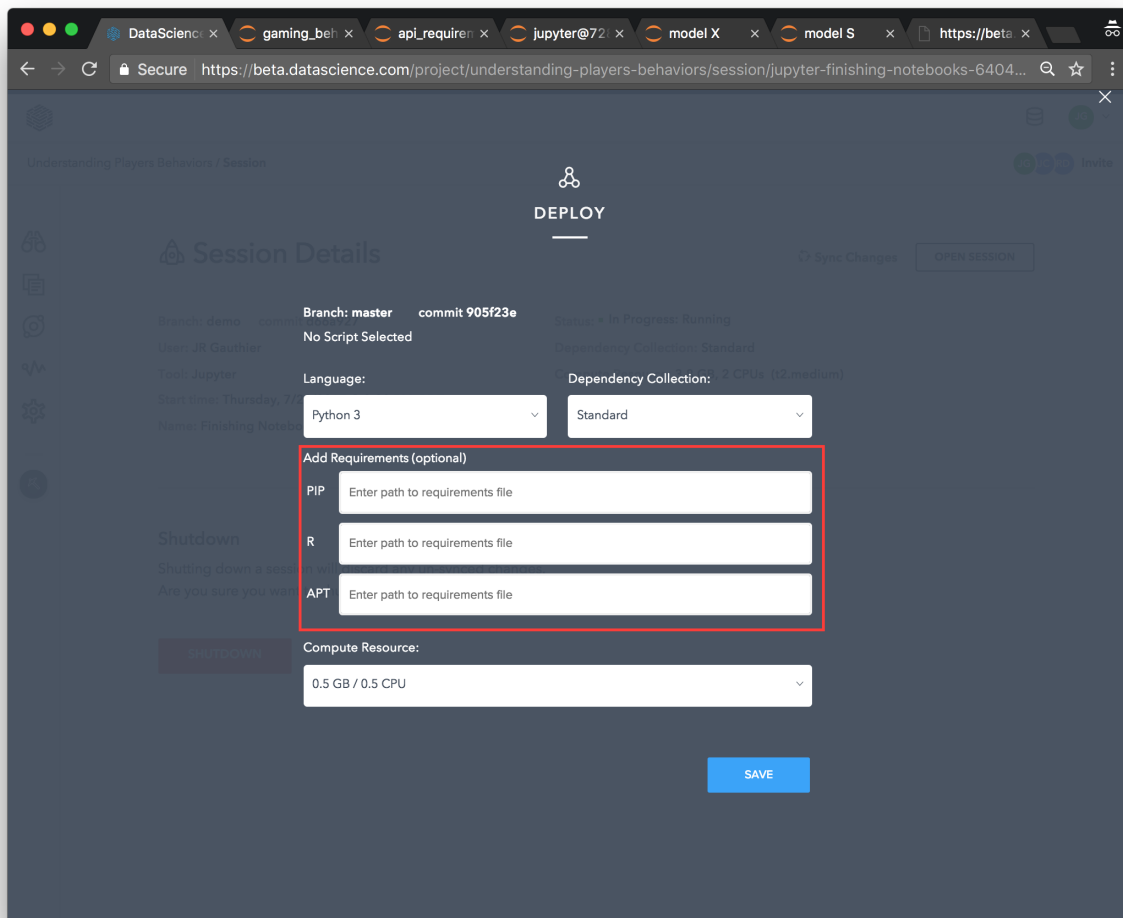
When deploying your model as a REST API, it is important that the API environment matches the one you used to develop the model. You achieve this by using dependency files. In the snapshot below we show where to put the names of the dependency files in the Deploy window.

#### 4.6.3.3 When Scheduling a Run

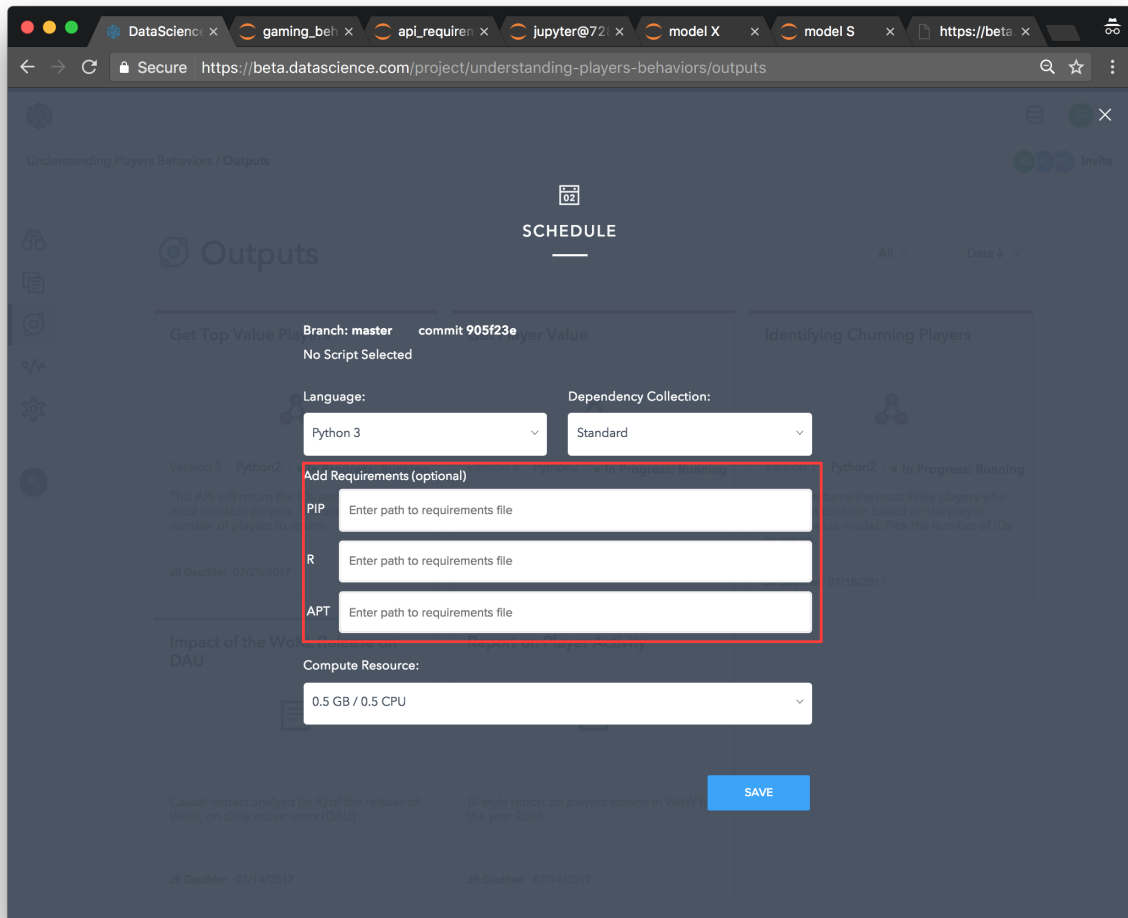
The same idea applies when scheduling a run. In the snapshot below you can see where the requirements files can be inserted.

### 4.6.4 General Tips and Best Practices

- Put your requirements in the top level folder of your project.
- Add the installer suffix to your dependency file names. For example: `requirements_pip.txt` , `requirements_r.txt` and `requirements_apt.txt`.







### 4.6.5 Additional References on Dependency Files

- [pip](#): A thorough introduction to Python pip requirements files
- [apt](#): Intro to apt and apt-get

---

## Working in a Session

---

Sessions combine interactive data science tools with packages and compute resources. Sessions are perfect for iterative analytical work, such as exploratory data analysis or feature engineering. The Platform currently supports Jupyter and RStudio, with Zeppelin coming soon.

When you launch a session, you may select from the the default set of environments created by your administrator. You can install additional libraries once inside a session, just like you would on a regular laptop (for example, using `pip` in Python). To learn more, see the [Environments](#) section.

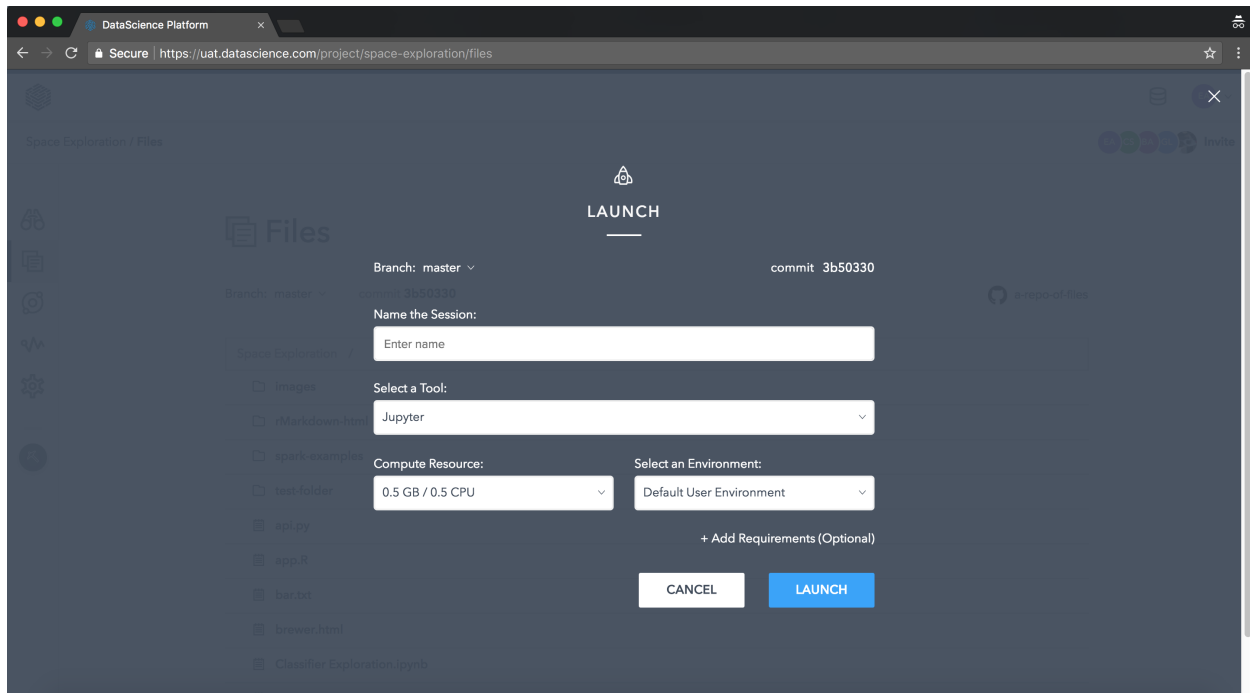
The DataScience.com Platform currently supports two interactive session tools:

- **Jupyter:** Jupyter is a staple in the Python open source data community, but has kernels for R and many other languages. For more resources, see the [Project Jupyter community page](#).
- **RStudio:** RStudio is a fully-featured development environment primarily for R programmers. The DataScience.com Platform supports the open source version of RStudio. For more information, see [their docs](#).

### 5.1 Launch a session

To start a session, select “Launch a Session” from the project actions button, then configure the following options:

- **Branch:** Determine the branch of your repo that you’ll work on. The files from the most recent commit on your branch will be available in the session.
- **Name:** Opt whether to name the session to help you keep track of multiple, concurrent sessions.
- **Tool:** Choose an interactive tool to use in your session.
- **Compute Resource:** Select from a list of machine sizes specified by your administrator.
- **Environment:** Choose a set of pre-installed libraries. For more on environments, see the [Environments and Dependencies page](#).
- **Additional Requirements:** Install additional dependencies at runtime from a text file. For more on additional requirements, see the [Environments and Dependencies page](#).



You can navigate back to a running session from your project's Activity tab, or from the Running Resources menu, shown here:

## 5.2 Sync changes

Just like traditional Git workflows on a personal computer, sessions clone from a branch, changes are staged (automatically by the sync menu), and then you push your changes with a commit message back to the Git remote.

After you've made some changes to your files in a session, you can save them by syncing back to the Git repo. From the top Platform chrome bar in your session, drop down the Session menu, and select Sync.

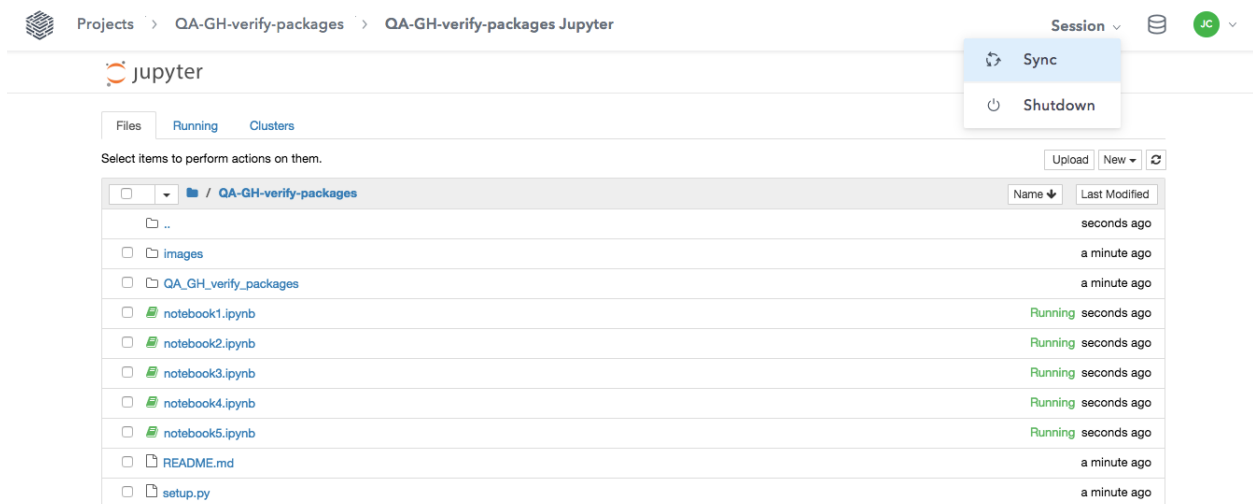
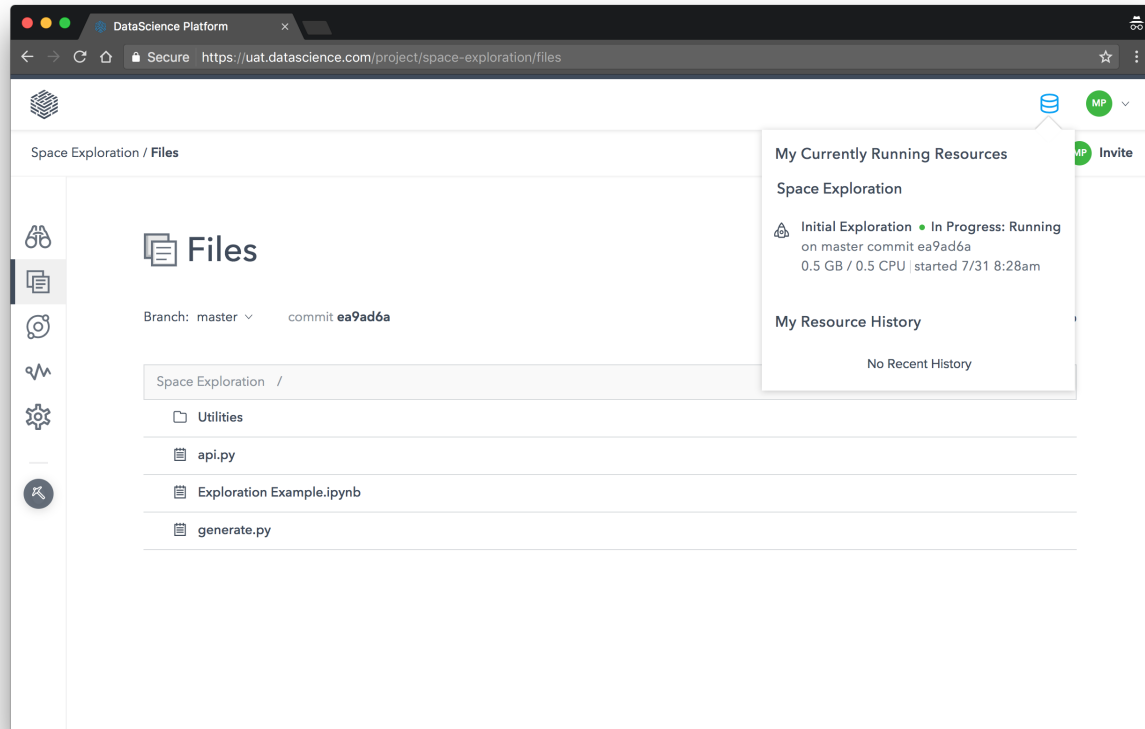
On the Sync menu, you'll see which files have been added, deleted, or modified. Using the checkboxes, you can select which files you would like to sync. You can enter an optional commit message and then sync your changes back to the Git repo.

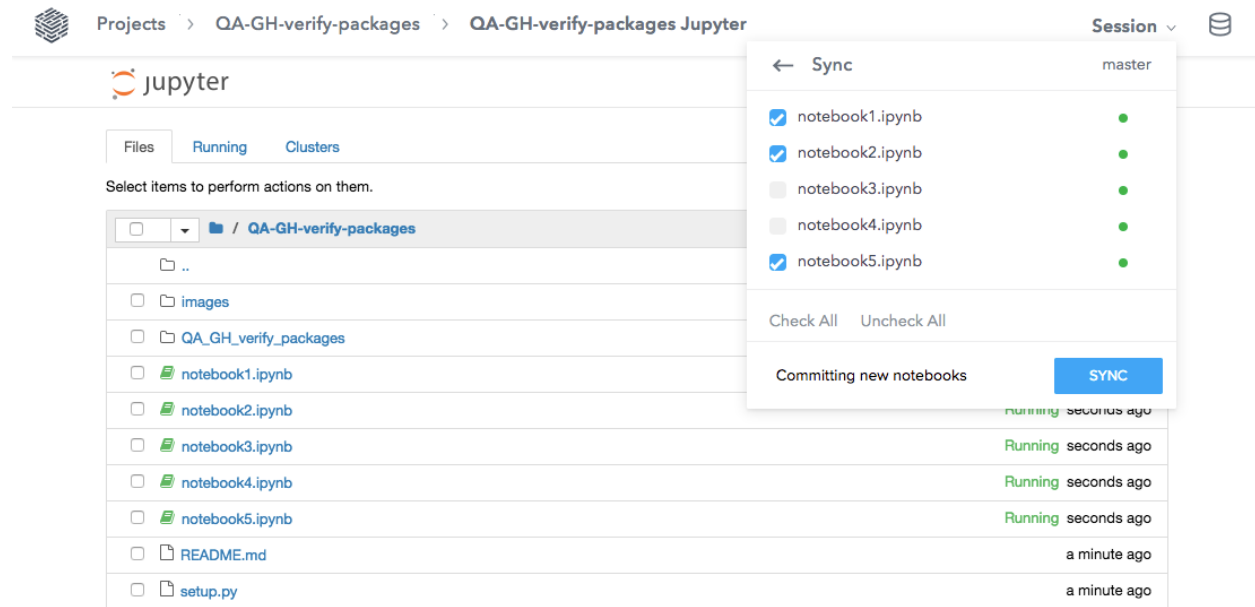
**Warning:** Be mindful of file sizes. Most Git providers have size limits for files you can store. For example, GitHub limits files to 100MB. Also, the DataScience.com Platform web app has a upload/download limit of 200MB, which affects downloading files from the Jupyter file browser.

If the file changes you've made don't conflict with changes your team has made since you started your session, the Platform will push all your files as a new commit to the active branch.

If there are conflicts, you'll have two choices:

- **Cancel:** this option reverts your Git status back to the moment you hit Sync. You may keep working and manually resolve conflicts using the Jupyter or RStudio file editors.
- **Create Branch:** this option creates a new branch and pushes your changes to that branch. The parent of the branch will be the commit that was originally loaded into your Session.





## 5.2.1 Git commands behind the scenes

Below are the exact commands that run for each Sync feature.

Loading the Sync menu:

```
git status
```

Sync action:

```
git add .
git commit -m <message you provide>
git fetch
git merge <branch you chose when launching> --no-commit --no-ff
```

Cancelling a Sync after a conflict:

```
git reset
```

Creating a new branch after a conflict:

```
git branch <name you provide>
```

## 5.3 Shut down a session

A session will run and consume compute resources until you stop it. To shut down a session, navigate to the Session menu in the top bar and select Shutdown.

**Warning:** You can't recover unsaved changes from a session after shutting down. If you want to save the work you have done, make sure to sync your files before shutting down.

---

## Connect to Data Sources

---

This section shows some common data connections used on the platform. We provide examples in both Python and R (when available).

### 6.1 AWS Redshift

This section shows how to connect Amazon Web Services (AWS) Redshift as a data source on the Platform.

#### 6.1.1 Python

```
# Platform Kernels: Python 2,3
# Libraries: psycopg2==2.7.1, pandas==0.20.3

# We adopt the psycopg2 client library to connect to
# postgresdb like redshift:
import psycopg2
import os
import pandas as pd

def RS_postgres_query(query_str, creds):
    """A sample query to validate the working of the db connection.

    Parameters
    -----
    query_str: string
        A string which contains the query you want to run.

    creds: dict
        Contains your RS/Postgres credentials. The dictionary should have
        five keys : "DATABASE_NAME", "REDSHIFT_PG_PORT", "REDSHIFT_PG_PASSWORD",
```

```
"REDSHIFT_PG_USER", "REDSHIFT_HOST". Those should be stored
as environment variables (os.environ[]) on the platform.
```

```
Returns
-----
```

```
A pandas dataframe with the results of the query.
```

```
"""
conn = psycopg2.connect(**creds)
return pd.read_sql(query_str, conn)
```

### 6.1.1.1 Usage Example

```
# Platform Kernels: Python 2,3
# Libraries: psycopg2==2.7.1, pandas==0.20.3

import psycopg2
import os
import pandas as pd
from redshift_postgres import RS_postgres_query

# Put the access credentials to the database in your environment variables
# into a dictionary.
creds = dict(database=os.environ['DATABASE_NAME'],
              port=os.environ['REDSHIFT_PG_PORT'],
              password=os.environ['REDSHIFT_PG_PASSWORD'],
              user=os.environ['REDSHIFT_PG_USER'],
              host=os.environ['REDSHIFT_PG_HOST'])

results_df = RS_postgres_query("""SELECT * FROM my_table""", creds)
```

### 6.1.2 R

```
# THIS IS NOT WORKING IN JUPYTER NOTEBOOKS

# options(repos=structure(c(CRAN="https://cran.cnr.berkeley.edu/")))

#install.packages("RPostgreSQL")
require("RPostgreSQL")

query_redshift <- function(conn, query){
  #' Queries redshift database with specified query
  #'
  #' Parameters
  #' -----
  #'
  #' conn: connection object
  #'
  #' query: string
  #'       SQL query to run
```



```
df_postgres <- dbGetQuery(conn, query)
return(df_postgres)

}
```

### 6.1.2.1 Usage Example

```
main <- function() {

# Make sure these are installed before running the code snippet below

## Install RJava
#!sudo apt-get update
#!sudo apt-get -y install default-jre
#!sudo apt-get -y install default-jdk
#!R CMD javareconf
#!sudo apt-get -y install r-cran-rjava

## Install driver for Postgres
#!sudo apt-get -y install libpq-dev

#install.packages("RPostgreSQL")
require("RPostgreSQL")

# loads the PostgreSQL driver
drv <- dbDriver("PostgreSQL")

# creates a connection to the postgres / redshift database
conn <- dbConnect(drv,
                   dbname = Sys.getenv("RS_DB"),
                   host = Sys.getenv("RS_HOST"),
                   port = Sys.getenv("RS_PORT"),
                   user = Sys.getenv("RS_USER"),
                   password = Sys.getenv("RS_PASSWORD"))

# define SQL query
query = "SELECT * FROM pg_catalog.pg_tables;"

# query redshift database
df_postgres = query_redshift(conn, query)
print(df_postgres)

}
```

## 6.2 AWS S3

This section shows how to connect Amazon Web Services (AWS) S3 as a data source on the Platform.

### 6.2.1 Python

```
# Platform Kernels: Python 2,3
# Libraries: boto3==1.4.4
```

```

import boto3
import os

def s3_ls(bucket_name, path, creds):
    """List contents of an S3 bucket specified in prefix 'path'

    Parameters
    -----

    bucket_name: string
        Name of the bucket of interest

    path: string
        Prefix of interest. Do not include a "/" to start.
        e.g. for s3://a-bucket/folderA/ => path = "folderA/"

    creds: dict
        Contains your AWS S3 credentials. The dictionary should have
        two keys : "S3_ACCESS_KEY" and "secret_key". Those should be stored
        as environment variables (os.envion[]) on the platform.

    Returns
    -----

    A list of the files

    """
    s3_client = boto3.client(service_name='s3',
                             aws_access_key_id=creds["AWS_ACCESS_KEY_ID"],
                             aws_secret_access_key=creds["AWS_SECRET_ACCESS_KEY"])

    if path != '' and path[-1] != '/':
        path += '/'

    files = []
    directories = []

    try:
        for fname in s3_client.list_objects(Bucket=bucket_name, Prefix=path)['Contents':
            if '/' not in fname['Key'].replace(path, ''):
                files.append(fname['Key'].replace(path, ''))
            elif fname['Key'].replace(path, '').split('/')[0] + '/' not in_
directories:
                directories.append(fname['Key'].replace(path, '').split('/')[0] + '/')

    except KeyError:
        return('Directory Not Found')

    return(directories + files)

def s3_pull_file(bucket_name, filepath, local_dir, creds):
    """Pull a file (any format) from S3 into the platform environment

```

After the file has been pulled in,  
it can be read into Python using the usual methods (e.g. `open()`)

Parameters  
-----

```
bucket_name: string
    Name of the bucket of interest

filepath: string
    File prefix of interest. Do not include a "/" to start.
    e.g. for s3://a-bucket/folderA/file.dat => filepath = "folderA/file.dat"

local_dir: string
    Local path where you want to store the file.
    e.g. local_dir = 'tmp_storage/file_tmp.dat'

creds: dict
    Contains your AWS S3 credentials. The dictionary should have
    two keys : "S3_ACCESS_KEY" and "S3_SECRET_KEY". Those should be stored
    as environment variables (os.environ[]) on the platform.

"""

s3_client = boto3.client(service_name='s3',
                        aws_access_key_id=creds["AWS_ACCESS_KEY_ID"],
                        aws_secret_access_key=creds["AWS_SECRET_ACCESS_KEY"])

# local values:
local_filename = os.path.basename(local_dir)
local_dirname = os.path.dirname(local_dir)

if not os.path.exists(local_dirname):
    os.makedirs(local_dirname)

# Download the file
s3_client.download_file(Bucket=bucket_name, Key=filepath, Filename=local_dir)

print("Your file is now available at {}".format(local_dir))

def s3_push_file(bucket_name, local_filepath, s3_filepath, creds):
    """Push a file from the platform environment into S3

    Parameters
    -----

    bucket_name: string
        Name of the bucket of interest

    local_filepath: string
        Local filepath of the file of interest (e.g. "/home/jupyter/data/filea.dat")

    s3_filepath: string
        prefix of the file to be stored on s3 (e.g. "docs/filea.dat")

    creds: dict
        Contains your AWS S3 credentials. The dictionary should have
```

*two keys : "S3\_ACCESS\_KEY" and "S3\_SECRET\_KEY". Those should be stored as environment variables (os.environ[]) on the platform.*

```

"""
s3_client = boto3.client(service_name='s3',
                        aws_access_key_id=creds["AWS_ACCESS_KEY_ID"],
                        aws_secret_access_key=creds["AWS_SECRET_ACCESS_KEY"])

try:
    s3_client.upload_file(local_filepath, bucket_name, s3_filepath)
    print("Uploaded to " + "s3://" + bucket_name + "/" + s3_filepath)
except BaseException as e:
    print("Upload error for " + local_filepath)
    print(str(e))

```

### 6.2.1.1 Usage Example

```

# Platform Kernels: Python 2,3
# Libraries: boto3==1.4.4

import boto3
import os
from s3 import s3_ls, s3_pull_file, s3_push_file

# Usage example:
creds = {"AWS_ACCESS_KEY_ID": os.environ["AWS_ACCESS_KEY_ID"],
        "AWS_SECRET_ACCESS_KEY": os.environ["AWS_SECRET_ACCESS_KEY"]}
bucket_name = os.environ["S3_BUCKETNAME"]
home_path = os.path.expanduser('~')

# List the content of bucket
list_of_files = s3_ls(bucket_name, "prefix1/", creds)

# Pull A file from S3:
s3_pull_file(bucket_name, 'prefix1/README', "{0}/tmp/README".format(home_path), creds)

# Push A file to S3:
s3_push_file(bucket_name, "{0}/tmp/README".format(home_path), "prefix2/README", creds)

```

### 6.2.2 R

```

# Platform Kernels: R3
# Libraries: aws.s3==0.3.3, utils=3.3.2

library('aws.s3')
library('utils')

set_S3_keys <- function(YOUR_ACCESS_KEY, YOUR_SECRET_KEY) {
  #' Sets the credentials as environment variables.
  #'
  #' ALL aws.s3 functions will look for your keys as environment variables

```

```

#' by default
#'
#' Parameters
#' -----
#'
#' YOUR_ACCESS_KEY: string
#'     AWS S3 access key
#'
#' YOUR_SECRET_KEY: string
#'     AWS S3 secret key

Sys.setenv('AWS_ACCESS_KEY_ID' = YOUR_ACCESS_KEY,
           'AWS_SECRET_ACCESS_KEY' = YOUR_SECRET_KEY)
}

s3_ls <- function(bucket_name, path){
  #' List contents of an S3 bucket specified in prefix 'path'
  #'
  #' Parameters
  #' -----
  #'
  #' bucket_name: string
  #' Name of the bucket of interest
  #'
  #' path: string
  #' Prefix of interest. Do not include a "/" to start.
  #' e.g. for s3://a-bucket/folderA/ => path = "folderA/"
  #'
  #' Returns
  #' -----
  #'
  #' A list of the files

  return(get_bucket(bucket_name, prefix = path))
}

s3_import_data <- function(bucket_name, filepath, read_func=NULL){
  #' Pull and import a file from S3 into memory
  #'
  #'
  #' Parameters
  #' -----
  #'
  #' bucket_name: string
  #'     Name of the bucket of interest
  #'
  #' filepath: string
  #'     File prefix of interest. Do not include a "/" to start.
  #'     e.g. for s3://a-bucket/folderA/file.dat => filepath = "folderA/file.dat"
  #'
  #' read_func: function
  #'     Define the function used to import the data. Determined by object file type.
  #'     Example readers:
  #'         - writes an RData (default)
  #'         - read.csv (csv)
  #'         - read.table (text)
  #'         - read.xls (excel)
  #'         - read.mtp (minitab)

```

```
#'      - read.spss (spss)
#'
```

```
if (is.null(read_func)){

  # Save an RData object to s3
  return(s3load(object = filepath, bucket = bucket_name))
} else {

  # Use write_func to save to s3
  return(s3read_using(FUN = read_func,
                      object = filepath,
                      bucket = bucket_name))
}
}
```

```
s3_export_data <- function(data, bucket_name, filepath, write_func=NULL){
  #' Push a object in memory to S3
  #'
  #'
  #' Parameters
  #' -----
  #'
  #' data: object
  #'      Variable containing data to push to S3
  #'
  #' bucket_name: string
  #'      Name of the bucket of interest
  #'
  #' filepath: string
  #'      File prefix of interest. Do not include a "/" to start.
  #'      e.g. for s3://a-bucket/folderA/file.dat => filepath = "folderA/file.dat"
  #'
  #' read_func: function
  #'      Define the function used to import the data. Determined by object file type.
  #'      Example readers:
  #'      - writes an RData (default)
  #'      - write.csv (csv)
  #'      - write.table (text)

  if (is.null(write_func)){

    # Save an RData object to s3
    s3save(data, bucket = bucket_name, object = filepath)
  } else {

    # Use write_func to save to s3
    s3write_using(data,
                  FUN = write_func,
                  object = filepath,
                  bucket = bucket_name)
  }
}
```

### 6.2.2.1 Usage Example

```
main <- function() {

  # The name of the bucket of interest:
  bucket_name <- Sys.getenv("S3_BUCKETNAME")

  set_S3_keys(YOUR_ACCESS_KEY = Sys.getenv("AWS_ACCESS_KEY_ID"),
              YOUR_SECRET_KEY = Sys.getenv("AWS_SECRET_ACCESS_KEY"))

  # List the content of a bucket with a specified prefix:
  s3_ls(bucket_name, path = 'prefix1/')

  # Importing

  # Pull a text file from S3:
  s3_import_data(bucket_name, 'prefix1/data.txt', read_func=read.table)

  # Pull a csv file from S3:
  s3_import_data(bucket_name, 'prefix1/data.csv', read_func=read.csv)

  # Pull RData file from S3:
  s3_import_data(bucket_name, 'prefix1/data.RData')

  # Exporting

  data <- "README"

  # Push a txt file to S3:
  s3_export_data(data, bucket_name, 'prefix2/data.txt', write_func=write.table)

  # Push a csv file to S3
  s3_export_data(data, bucket_name, 'prefix2/data.csv', write_func=write.csv)

  # Push a RData file to S3:
  s3_export_data(data, bucket_name, 'prefix2/data.RData')

}
```

## 6.3 MySQL

This section shows how to connect MySQL as a data source on the Platform.

### 6.3.1 Python

```
# Platform Kernels: Python 2,3
# Libraries: sqlalchemy==1.1.11, MySQLdb==1.2.5, pandas==0.20.3

# Make sure these are installed before running the code snippet below :
#!sudo apt-get -y install libmysqlclient-dev
#!pip install MySQL-python
#!pip install sqlalchemy

from sqlalchemy import create_engine
```

```
import MySQLdb as mdb
import pandas as pd

def pull_data_from_mysql(db(query, creds):
    """Given a SQL query and a connection object, this function
    returns a pandas dataframe with the results of the query.

    Parameters
    -----

    query: string
        SQL query of interest

    creds: dict
        Contains your mysql database credentials. The dictionary should have
        four keys : "MYSQL_HOST", "MYSQL_USER", "MYSQL_PASSWD", "MYSQL_DB".
        Those should be stored as environment variables (os.environ[])
        on the platform.

    Returns
    -----

    a pandas dataframe with the results of the query.

    """
    # Establishing a connection object to the MySQL Database
    conn = mdb.connect(creds['MYSQL_HOST'], creds['MYSQL_USER'],
                       creds['MYSQL_PASSWD'], creds['MYSQL_DB'])
    return pd.read_sql(query, conn)

def insert_data_into_mysql(db(dataframe, creds, table, mode='append'):
    """Either appends, replace or fail the content of a dataframe
    to a MySQL table.

    Parameters
    -----

    dataframe: pandas dataframe
        Dataframe of interest

    creds: dict
        Credentials to access the MySQL database.

    table: string
        Table in database that you would like to modify

    mode: string
        Database writing. Possible values are :
        - fail: if table exists, do nothing
        - replace: if table exists, drop it, recreate it, and insert data
        - append: if table exists, insert data. Create table if does not exist.
        see pandas.DataFrame.to_sql() for more info.

    """
```



```
# Define string for creating engine
eng_str = 'mysql+mysqldb://{0}:{1}@{2}/{3}'.format(creds['MYSQL_USER'],
                                                creds['MYSQL_PASSWD'],
                                                creds['MYSQL_HOST'],
                                                creds['MYSQL_DB'])

engine = create_engine(eng_str, echo=False)

dataframe.to_sql(name='{0}'.format(table),
                 con=engine, if_exists=mode, index=False)
```

### 6.3.1.1 Usage Example

```
# Platform Kernels: Python 2,3
# Libraries: sqlalchemy==1.1.11, MySQLdb==1.2.5, pandas==0.20.3

# Make sure these are installed before running the code snippet below :
#!sudo apt-get -y install libmysqlclient-dev
#!pip install MySQL-python
#!pip install sqlalchemy

from sqlalchemy import create_engine
import MySQLdb as mdb
import pandas as pd

from mysql import pull_data_from_mysqldb, insert_data_into_mysqldb

# Usage Example :

# Put in the credentials in a dictionary:
creds = dict(MYSQL_HOST=os.environ['MYSQL_HOST'],
             MYSQL_USER=os.environ['MYSQL_USER'],
             MYSQL_PASSWD=os.environ['MYSQL_PASSWD'],
             MYSQL_DB=os.environ['MYSQL_DB'])

# Pull data from a MySQL database:
data = pull_data_from_mysqldb("SELECT * FROM my_table", creds)

# Define dataframe to append
df = pd.DataFrame({'customer_id': [1, 2, 3, 4, 5],
                  'height': [2, 3, 4, 5, 6]})

# Push a dataframe to a MySQL database:
insert_data_into_mysqldb(df, creds, 'my_table', mode='replace')
```

### 6.3.2 R

```
# Platform Kernels: R3
# Libraries: RMySQL==0.10.13, DBI==0.6-1

## Install MySQL
#!sudo apt-get -y update
#!sudo apt-get -y install libmysqlclient-dev
```

```

# Install and import RMySQL
install.packages("RMySQL")

library(RMySQL)
# RMySQL_0.10.13 DBI_0.6-1

readFromSQL <- function(conn, query) {
  # Read from a MySQL database and returns the results
  # of the query as a dataframe.
  #
  # Args:
  #   query: String containing the SQL query.
  #   conn: connection object from dbConnect()
  #
  # Returns:
  #   dataframe with the results of the query.
  #
  resultsDataframe <- dbGetQuery(conn = conn, statement = query)
  return(resultsDataframe)
}

writeToSQL <- function(conn, tableName, dataframe, mode='append') {
  # Write to a SQL database table the dataframe.
  #
  # Args:
  #   dataframe: dataframe you want to write to a table.
  #   tableName: name of the table you want to write the data to.
  #   conn: connection object from dbConnect()
  #   mode: write mode. Values are :
  #         - overwrite: Overwrite an existing table.
  #         - append: Append to an existing table.
  #
  if(mode=='overwrite'){
    dbWriteTable(conn = conn, name = tableName, value = dataframe, overwrite = TRUE)
  } else if(mode=='append'){
    dbWriteTable(conn = conn, name = tableName, value = dataframe, append = TRUE)
  } else
    print('Wrong choice of mode. Values are overwrite or append.')
}

```

### 6.3.2.1 Usage Example

```

main <- function() {

  # Install and import RMySQL
  # install.packages("RMySQL")

  library(RMySQL)

  # Set up the connection object:
  conn = dbConnect(MySQL(),
                    user=Sys.getenv("MYSQL_USER"),
                    password=Sys.getenv("MYSQL_PASSWD"),

```

```

        host=sys.getenv("MYSQL_HOST"),
        dbname=sys.getenv("MYSQL_DB"))

# Read from a MySQL database table:
dataframeResults <- readFromSQL(conn, "SELECT * FROM my_table")

# Write a dataframe to a MySQL database table:
writeToSQL(conn, "my_table", dataframeResults , mode='overwrite')

# Close connection
dbDisconnect(conn)
}

```

## 6.4 Google BigQuery

This section shows how to connect to Google BigQuery as a data source on the Platform.

### 6.4.1 Python

```

# Platform Kernels: Python 2,3
# Snippet Libraries: google.cloud.bigquery==0.26.0, pandas==0.20.3

# Installation/upgrade of google.cloud.bigquery :
# !pip install --upgrade google-cloud-bigquery

import os
import pandas as pd
from google.cloud import bigquery
from google.cloud.bigquery import SchemaField

def pull_data_from_bigquery(query_string):
    """ Pull data using Google BigQuery.

    Parameters
    -----

    query_string: string
        Your BigQuery query string.

    Returns
    -----

    Pandas DataFrame with the results of the query.

    Note:
    Assumes the following environment variables
        - GOOGLE_CLOUD_PROJECT set to the BigQuery Project ID
        - GOOGLE_CLOUD_CRED_PATH set to GCloud credential JSON file

    """
    # More details here : https://google-cloud-python.readthedocs.io/en/latest/core/modules.html
    ↪modules.html

```

```
client = bigquery.Client.from_service_account_json(os.environ['GOOGLE_CLOUD_CRED_
↪PATH'])

query = bigquery.query.QueryResults(query_string, client)
query.run()
assert query.complete, 'Query not completed'
dat = query.fetch_data()
df = pd.DataFrame(dat.query_result.rows,
                  columns=[x.name for x in dat.query_result.schema])

return df
```

### 6.4.1.1 Usage Example

```
from bigquery import pull_data_from_bigquery

# Usage example:

# In your environment file:
# export GOOGLE_CLOUD_PROJECT='your-project-id'
# export GOOGLE_CLOUD_CRED_PATH='path/to/credentials/file.json'

query = """SELECT year, month, day, weight_pounds
            FROM [publicdata:samples.natality]
            LIMIT 5"""
results = pull_data_from_bigquery(query)
```

## 6.4.2 R

```
# RUN this in ** RStudio **

# Platform Kernels: RStudio
# Snippet Libraries: bigrquery==0.3.0, dplyr==0.5.0
#
# For more details on bigrquery, go to :
# https://github.com/rstats-db/bigrquery

# install the libraries :
install.packages("dplyr")
install.packages("bigrquery")
install.packages('httpuv')

library("bigrquery")

queryBigQuery <- function(project, queryString) {
  # Query BigQuery and returns the results of the query
  # as a data.frame.
  #
  # Args:
  #   project: project ID
  #   queryString: BigQuery query string
  #
  # Returns:
  #   dataframe of the query results.
```

```
#
queryResults <- query_exec(queryString, project=project)

return(queryResults)
}
```

#### 6.4.2.1 Usage Example

```
main <- function(){
# Usage Example:

library("bigrquery")

# Authenticate based on JSON token
set_service_token(Sys.getenv('GOOGLE_CLOUD_CRED_PATH'))

# Use your project ID here
project <- Sys.getenv('GOOGLE_CLOUD_PROJECT') # put your project ID here

# Querying BigQuery's public datasets :
queryString <- "SELECT year, month, day, weight_pounds FROM [publicdata:samples.
↪natality] LIMIT 5"

queryResults <- queryBigQuery(project, queryString)
}
```

## 6.5 SAP-HANA

Yes, you can freely query SAP HANA. You can use HANA Studio as well as the Python ODBC bridge to access SAP HANA data within our Platform. You may also query SAP HANA data using R or Scala. These connection methods, packages, and resources can be built into a standardized, reusable environment making HANA data access quick, simple, and reliable.



---

## Scripts and Scheduled Runs

---

Like other features on the platform (Sessions, APIs, etc.), you can run scripts in isolated containers with dedicated server resources. The Runs and Scheduled Runs features let you execute code, view logs, and collect outputs.

### 7.1 Run a script

There are two ways to run a script. You can either navigate to a file's preview page and click the Run button in the top right, or use the Run a Script option in the Actions menu. Select the script to run by typing the path to the file and use the autocomplete feature. Configure the script run by choosing hardware size, language, environment, and a repo branch.

You can check the status of your run in two places. You'll find your run in the Currently Running dropdown in the top menu. Or, you can navigate to your Run Details page from the Activity Feed.

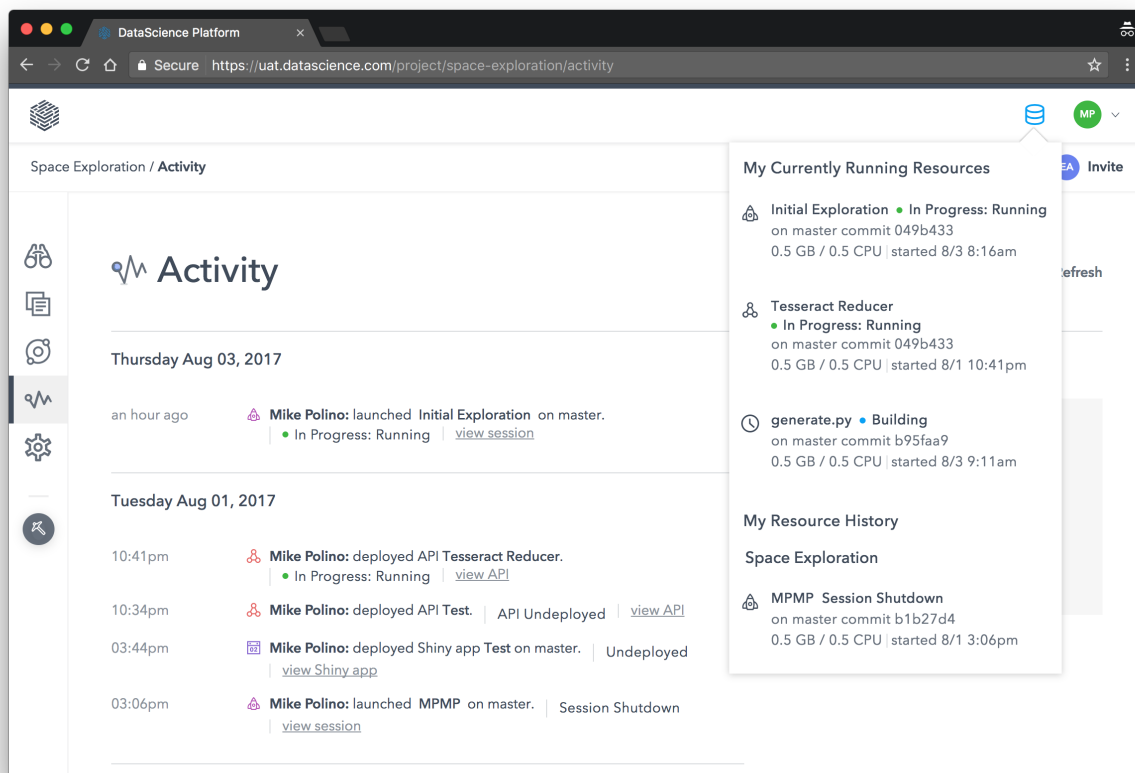
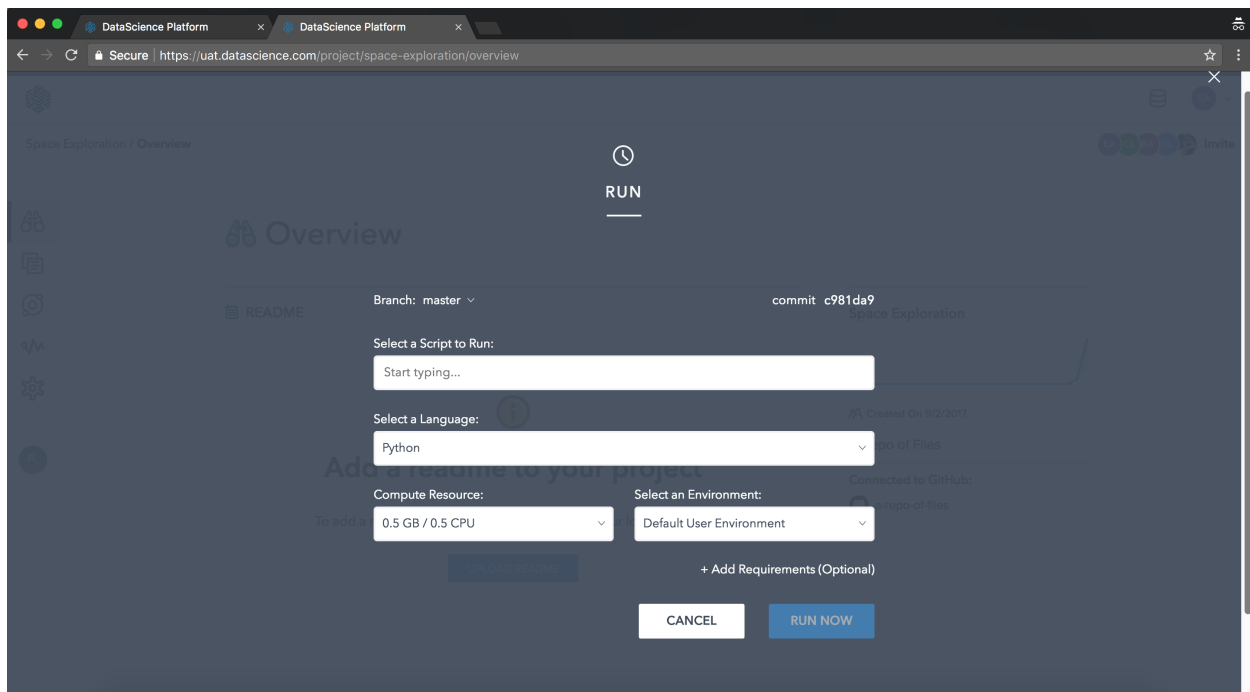
### 7.2 The Run Details page

The Run Details page shows the configuration of the container that ran the script, as well as the process's Standard Out and Standard Error. Standard Out will display anything that was printed by the script to `stdout`. Standard Error shows anything printed by the script to `stderr`.

### 7.3 Schedule a run

You can schedule scripts to run at regular intervals. To schedule a script, pick the Schedule A Script option from the Actions menu and select the file you wish to schedule. You'll find the same environment configuration options here, with the addition of the schedule timing.

To specify how often the script should run, you can choose from a set of standard options (hourly, daily, weekly, monthly) or specify a custom schedule.





The screenshot shows the 'Run Details' page for a script run. The page is titled 'Run Details' and includes a 'RERUN' button. The run information is as follows:

- Branch: run-script-test commit 09dcea4
- User: Elena Albright
- Script: script.py
- Additional Dependencies: none
- Start Time: Tuesday, 9/5/2017 @ 5:28 PM, PDT
- Status: Run Finished Successfully
- Language: Python
- Environment: default-user-environment
- Compute Resource: 0.5 GB, 0.5 CPUs

Below the run information, there are two expandable sections: 'Standard Out' and 'Standard Error', each with a '+' icon to expand the content.

The screenshot shows the 'Run Details' page with a scheduling overlay. The overlay includes the following fields and options:

- Enter schedule name...
- Select a Script to Run: Start typing...
- Select a Language: Python
- Compute Resource: 0.5 GB / 0.5 CPU
- Select an Environment: Default User Environment
- + Add Requirements (Optional)
- Run: Weekly
- On: M T W TH F SA SU
- At: 12 : 00 am | pm PDT
- CANCEL
- SCHEDULE

The background shows the 'Run Details' page with the same run information as the first screenshot.

### 7.3.1 Custom schedules

Aside from the Hourly, Daily, Weekly, and Monthly options, you can define a custom cron syntax schedule.

The custom schedule syntax can have five or six arguments separated by spaces. Each argument corresponds to the following:

- **Seconds (optional):** 0-59 (e.g., 10 would run 10 seconds after the start of a minute)
- **Minutes:** 0-59
- **Hours:** 0-23
- **Day of Month:** 1-31
- **Months:** 0-11 (note that January is 0)
- **Day of Week:** 0-6 (note that Sunday is 0)

Leaving out Seconds will default to running at 0 seconds. You can also replace any of the arguments with `*`, which means run no matter what the value is. For example, `30 15 10 5 1 *` will run at 10:15 am and 30 seconds, on February 5th, no matter what day of the week.

You can also use commas to specify lists of values, dashes to specify ranges, and slashes to specify increments. See below for more examples, and here for official documentation on [node-cron](#).

- `30 15 10 5 1 \*` - Run at 10:15:30 am on February 5th, no matter what day of the week
- `30 15 10 5 1 2` - Run at 10:15:30 am on February 5th, but only if it is a Tuesday
- `0 0 \* 0-3 6` - Run at midnight on every Saturday in January through April
- `0 0 \* 2,4 6` - Run at midnight on every Saturday in March and July
- `0 */30 15 * \*` - Run every 30 minutes on the 15th of every month, no matter what day of the week

## 7.4 Schedule Details page

You can reach a job's Schedule Details page from the Activity feed. This page shows you all past runs (with links to the Run Details pages), and lets you edit the frequency of the job. To edit anything other than the frequency (e.g., the hardware for the container), you need to create a new job. You may also stop all future runs for the job from the Schedule Details page.

The screenshot shows a web browser window with the URL `https://uat.datascience.com/project/space-exploration/job/generative-model-260264`. The page title is "Space Exploration / Job". The main content area is titled "Schedule Details" with a sub-header "Generative Model | Built Successfully | Active: ✓ Yes". Below this, it states "Runs at 05 minutes past the hour". The page is divided into two columns of metadata:

Metadata	Value
Branch:	master
commit:	b95faa9
Language:	Python 2
Script:	generate.py
Dependency Collection:	Standard
Last Edited By:	Mike Polino
Compute Resource:	0.5 GB, 0.5 CPUs

Below the metadata is a "Build Log" section with a "+" icon to expand it. At the bottom, there is a "Completed Runs:" section with a table showing the status of the last run.

Start time	Status
Thursday, 8/3/2017 @ 10:05 AM, PDT	Run Finished Successfully



Reports make sharing ideas easy and let you avoid copy/pasting results into emails or slides. With a single click, you can present your text, code, and charts from notebooks and then easily share with others in your company by sending them a link.

## 8.1 Publish a Report

You can publish the following file types as a report:

- .ipynb
- .md
- .html (e.g. from RStudio's knit function)

There are two ways to publish a report. From a file preview page for the file you wish to publish, click the Publish button above the file content. Or, use the Publish a Report option in the project actions dropdown. On the publishing menu displayed, fill in the path of your file and give the report a title and description.

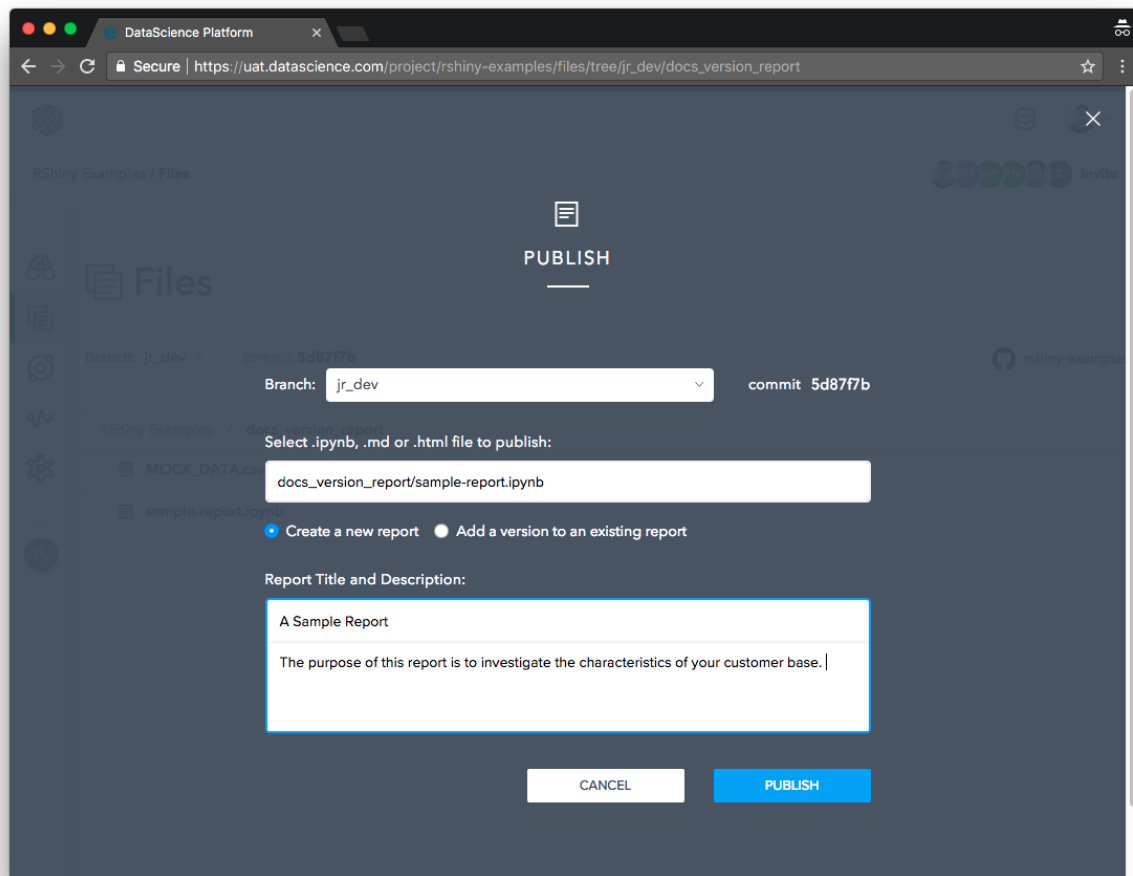
### 8.1.1 Preparing an .rmd File for Publishing

R Markdown files only save their outputs (like charts and tables) in a running RStudio session rather than saving outputs to the .rmd file. To publish an R Markdown document as a report on the Platform, first convert it to an HTML document inside your RStudio session.

From RStudio, run the R Markdown file from top to bottom, loading all the charts, tables, or other outputs into the view. Then in the RStudio console, run the following command:

```
rmarkdown::render('my-analysis.Rmd')
```

At a minimum, the header of the R Markdown should set the output type as HTML. Here's an example header:



```

---
title: "My Analysis"
output: html_document
---

```

The resulting HTML file should be synced back to the project's repository. Then from the Files tab or the actions dropdown, publish the HTML file as a report. You may choose to version your HTML files in the Git repo or delete them after publishing.

To control whether your audience sees the code for each cell of an R Markdown document, use the `echo` option, documented [here](#). For example, to prevent code from making it into the report, structure your code cells like this:

```

### Here's where my analysis gets interesting.

'''{r echo=FALSE}
plot(some_data)
'''
```

## 8.2 View and Manage a Report

Visit your report from the project Outputs page. On the Report page, you can remove a report with the Delete button, located above the report content.

Before sharing a link to a report with teammates, first make sure they're a member of the project. The View permission level is appropriate for teammates who are visiting a project just to review your reports.

## 8.3 Report Versions

Report versions allow you to make changes to an existing report. Instead of creating a whole new report, you can choose to simply bump up the version number of that report. This is particularly useful when data is being updated and you want to re-run the notebook and update the charts.

When looking at the report itself, you can see the version list in the Version dropdown menu on the left. Here you can switch between different report versions. In the top right, you also see different options under Delete: delete Current Version or All Versions. You can also publish a new version of the report by clicking on Publish New Version. There are two ways to create a new version of a report, discussed in more detail in the following sections.

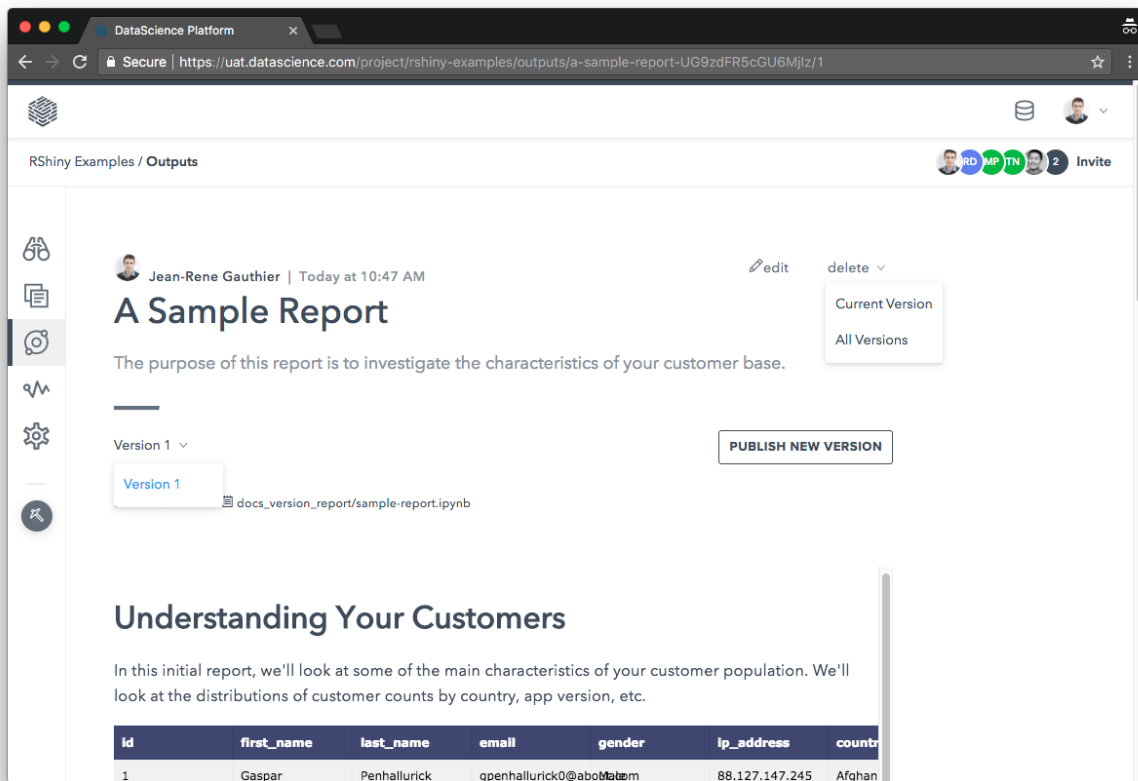
### 8.3.1 Publishing a New Version from Within a Report

Exemplified in the snapshot above, the first way to create a new version of a report is to go in the report itself and click Publish New Version. As a result, the following window will appear:

Double check the branch and commit ID. Make sure they are the ones you want to capture in your new report version. If you go back to the report, you should see that Version 2 is the one being used.

### 8.3.2 Publishing a New Version from the Action Button

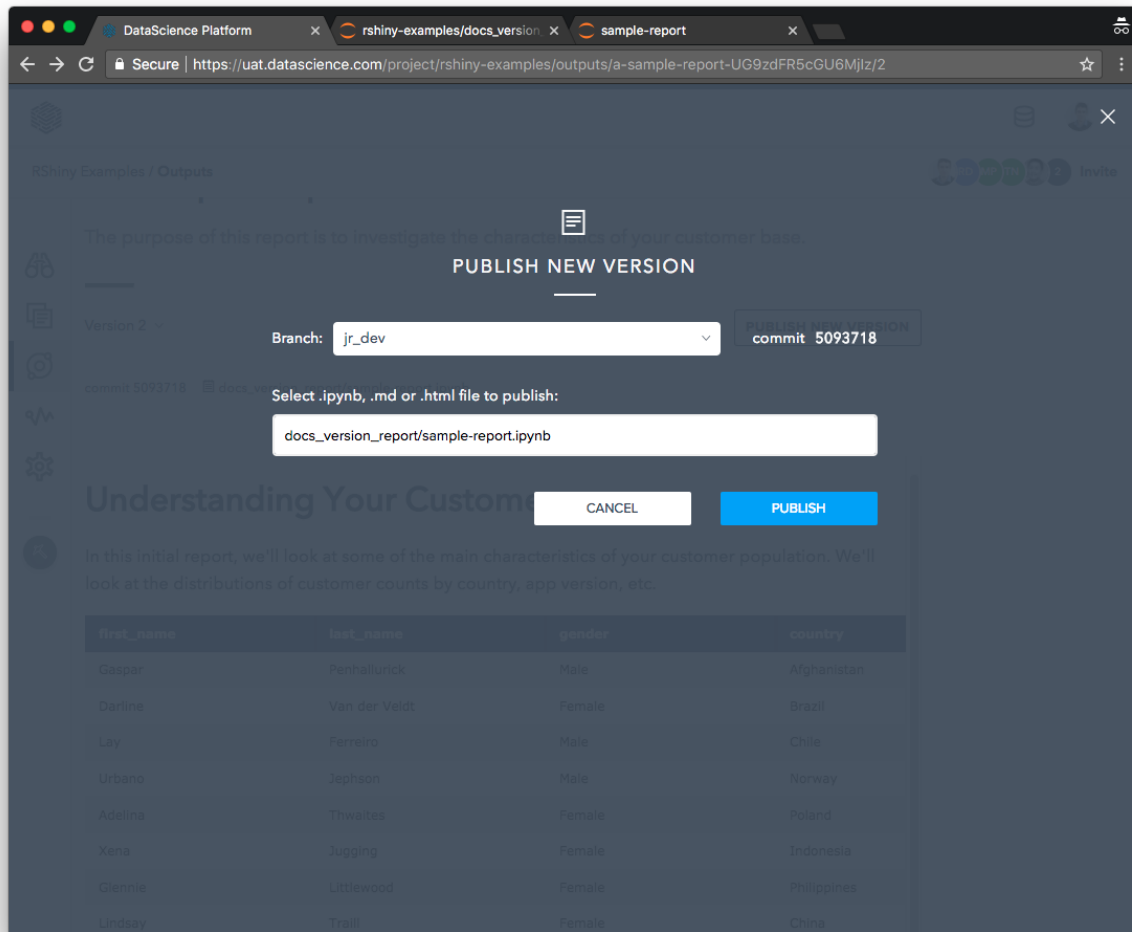
You can also use the action button to add a version to an existing report. Choose the Publish a Report option. Within that window, choose your branch and select the option stating "Add a version to an existing report." Write the path of your notebook and enter the name of your report. After clicking Publish, a new version of your report will appear.

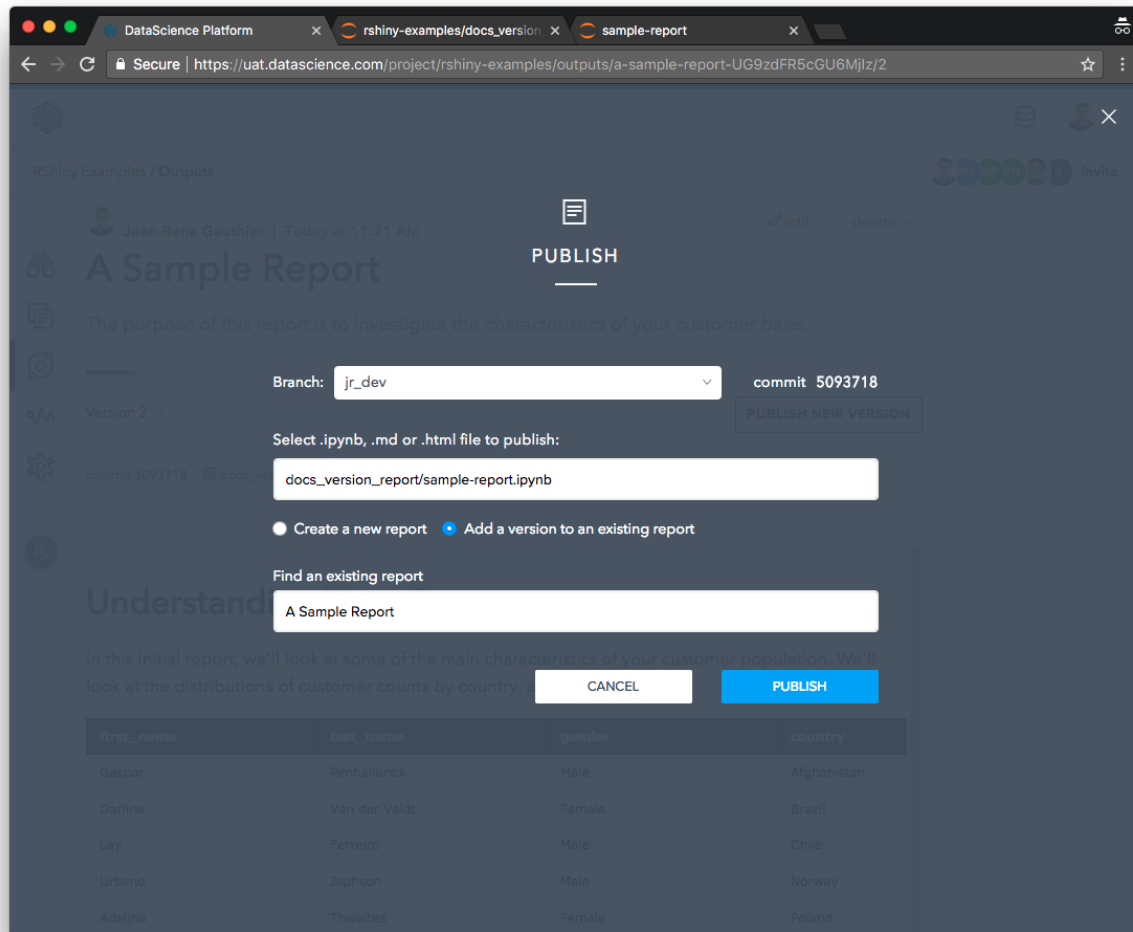


The screenshot shows the DataScience Platform interface. The browser address bar indicates a secure connection to <https://uat.datascience.com/project/rshiny-examples/outputs/a-sample-report-UG9zdFR5cGU6MjJz/1>. The page title is "RShiny Examples / Outputs". The report is titled "A Sample Report" and is authored by Jean-Rene Gauthier, dated "Today at 10:47 AM". The report content includes a description: "The purpose of this report is to investigate the characteristics of your customer base." Below the description, there is a "Version 1" dropdown menu and a "PUBLISH NEW VERSION" button. The report content also includes a section titled "Understanding Your Customers" with a paragraph: "In this initial report, we'll look at some of the main characteristics of your customer population. We'll look at the distributions of customer counts by country, app version, etc." Below this paragraph is a table with the following data:

id	first_name	last_name	email	gender	ip_address	country
1	Gaspar	Penhallurick	gpenhallurick0@abott.com	Male	88.127.147.245	Afghanistan

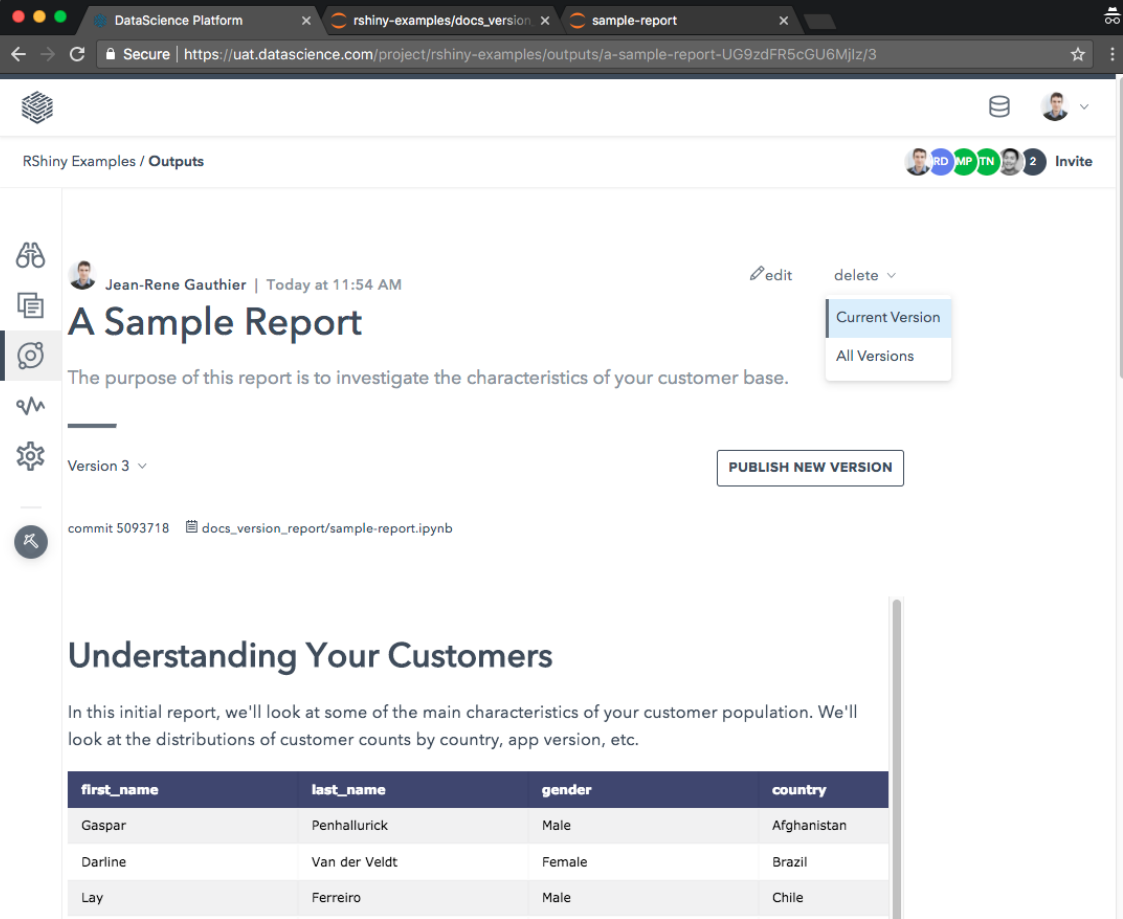






## 8.4 Delete a version

You can also delete a version of your report. Simply go in your report and choose either the Current Version or All Versions option under the Delete dropdown.



The screenshot shows the DataScience Platform interface. The browser address bar displays the URL: `https://uat.datascience.com/project/rshiny-examples/outputs/a-sample-report-UG9zdFR5cGU6Mjlz/3`. The page title is "RShiny Examples / Outputs". The report is titled "A Sample Report" by Jean-Rene Gauthier, dated "Today at 11:54 AM". The report content includes a description: "The purpose of this report is to investigate the characteristics of your customer base." and a section titled "Understanding Your Customers" with the text: "In this initial report, we'll look at some of the main characteristics of your customer population. We'll look at the distributions of customer counts by country, app version, etc." Below the text is a table with customer data.

first_name	last_name	gender	country
Gaspar	Penhallurick	Male	Afghanistan
Darline	Van der Veldt	Female	Brazil
Lay	Ferreiro	Male	Chile



---

## R Shiny Dashboards

---

Shiny by RStudio is a framework for turning R code into interactive dashboards. On the DataScience.com Platform, Shiny dashboards are deployed in dedicated containers and are accessible through a project's Outputs page.

### 9.1 Publish a dashboard

To publish a Shiny Application, select “Publish a Shiny App” from the actions menu and enter the following information to complete configuration:

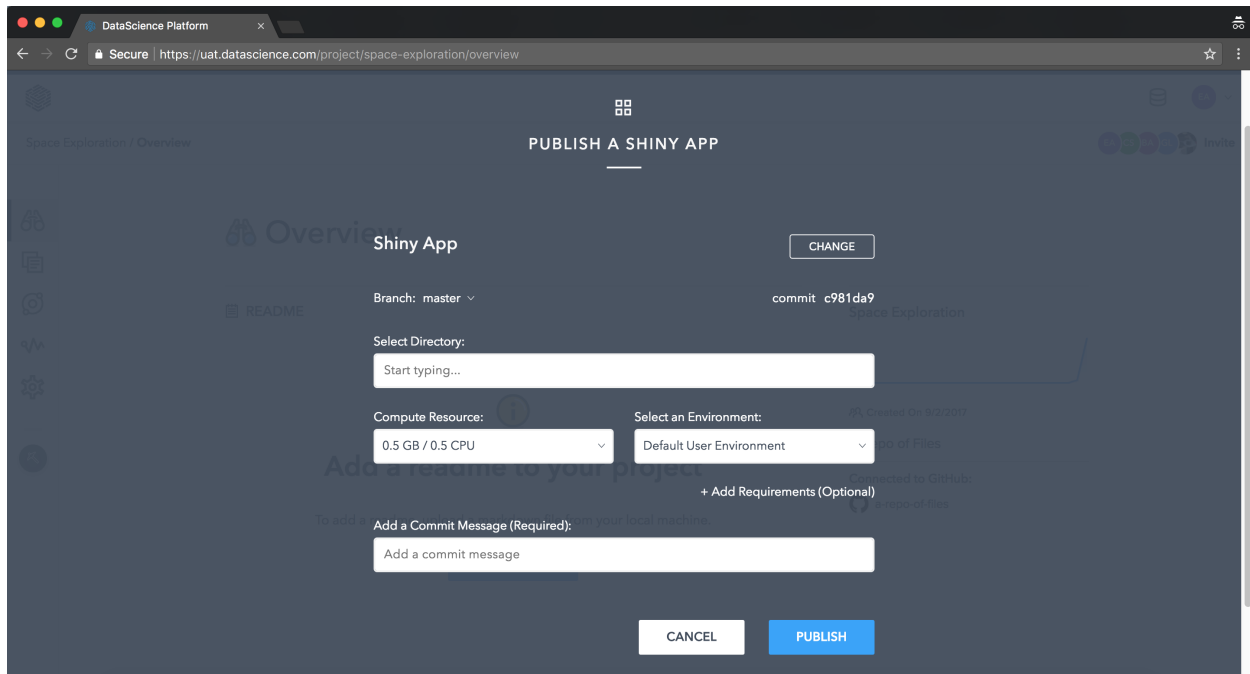
- **Name and description:** Enter your app's name (limit 55 characters) and description (limit 140 characters). These will be displayed on the project's Output page to help users find your app.
- **Branch and App Path:** See Running a Directory below for more information.
- **Compute Resource:** Select the size of the hardware where your app will be hosted.
- **Environment and Additional Requirements:** Determine the environment and any additional dependencies that are needed for your app to run.
- **Commit message:** Enter a short message to help identify this version of the app.

#### 9.1.1 Running a Directory

Shiny dashboards are served out of a directory in your project that contains all the files the dashboard needs to run.

To deploy a Shiny dashboard, specify a folder in your project with all of your Shiny app code, including the dashboard's entry-point file. Use the convention `app.R` for this entry-point file, which is where the core logic of the app lives. You may use a single entry-point file to build your dashboard or break it out into `modules`.

**Warning:** You must specify a directory, not a `.R` file when publishing a Shiny dashboard. Your dashboard won't have access to any files or folders higher in the project tree than the one you specified when publishing.



You can publish a new version of an existing dashboard by visiting that dashboard's page on the Outputs tab, then clicking the Publish New Version button. In this form, you can enter a new directory for your app code and include a message to let teammates know what changed.

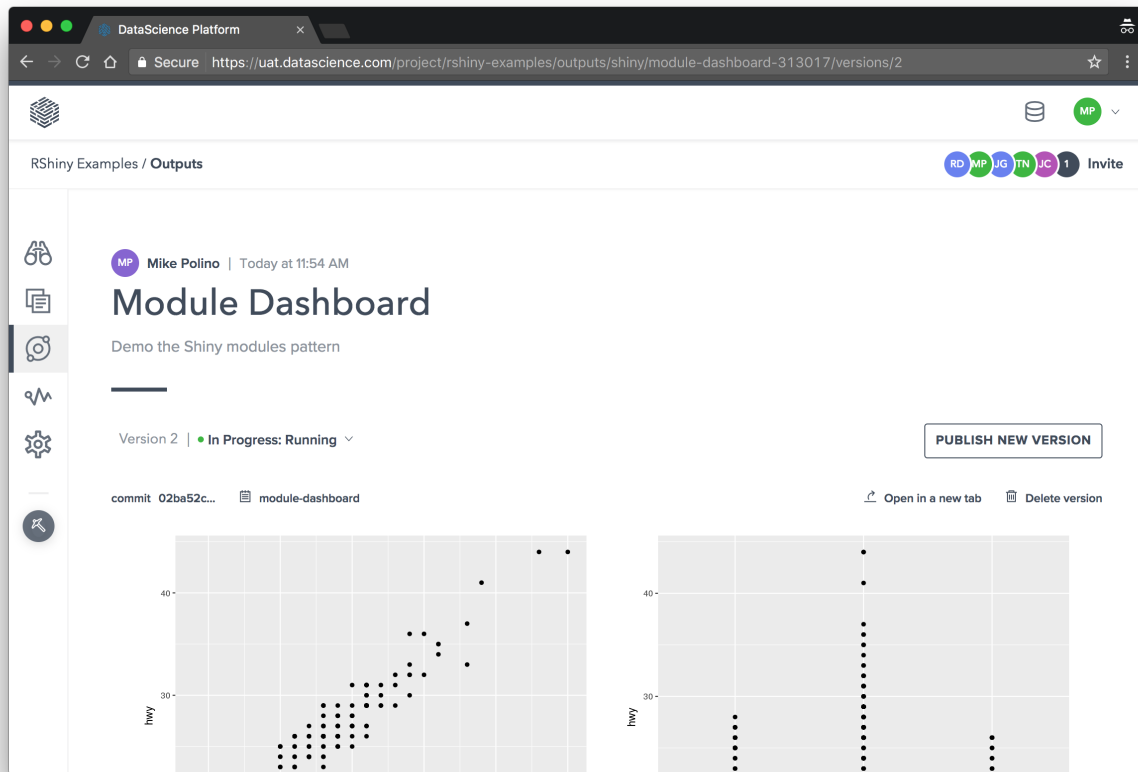
## 9.2 View and manage a dashboard

Shiny dashboards are viewable through the project Outputs tab. From this page, you can browse, publish, and delete versions of a dashboard. To share a dashboard in full-screen mode, use the Open in a new tab option above the dashboard content.

To share a link to a Shiny dashboard with a teammate, first invite them to the project.

You can find the deployed status of a dashboard under the versions dropdown, just above the dashboard content. Your dashboard is healthy when its status is `In Progress: Running`. If your application code has errors, the version status will change to `In Progress: Error`.

You can delete versions of your dashboard from this page. Deleting the last version of a dashboard deletes the entire dashboard page.







The Model APIs feature let you turn Python and R models into API endpoints that can be called by any app or service that can make web requests. APIs open up your models to real-time, integrated use cases. For example, a Model API could power a customer-facing app or back an internal dashboard at your company.

## 10.1 Overview

### 10.1.1 Deploy an API

When you deploy an API, a dedicated service is created with your project code and dependencies, and you specify the script and function that powers the endpoint. Your model will accept JSON object payloads as inputs; those values are passed to your function, and then the results are returned back to the requestor. For example, if you deployed this function:

```
def addXY(x, y):  
    return x + y
```

then calling the API with `{ 'x': 2, 'y': 3 }` will return 5.

To deploy a model API, select “Deploy an API” from the actions menu and enter the following information to complete configuration:

- **Name and description:** Enter your API’s name (limit 55 characters) and description (limit 140 characters). These will be displayed on the project’s Output page to help users find your API.
- **Branch:** Indicate the branch of your repo that will be loaded into the deployed API’s container. The entire repo will be loaded into the container, not just the file to be deployed.
- **Model Path:** Enter the name of the file that contains the function to be deployed.
- **Language:** Select the language that the model is written in.
- **Compute Resource:** Choose the size of the hardware where your API will be deployed.

- **Environment and Additional Requirements:** Determine the environment and any additional dependencies that are needed for your model to run.
- **The function:** Fill in the specific function (which must be in the file you chose above) that you wish to deploy. This function will take inputs from each incoming API request.
- **Example data:** Once your API is deployed, it will have an API page where users can view information about it. That page includes a section for trying sample data and seeing what the API would return with those inputs. If you supply example data in this step, it will be populated in the sample data section of your API page by default. Your sample data must be a valid JSON object.
- **Commit message:** A short message to help identify this version of the API.

**Warning:** When you deploy a script, the file must be in the root of the project. Deploying files in sub-folders is currently not supported.

**Warning:** Per good engineering practices, ensure there are no spaces in your file or folder names.

**Warning:** Your function's output is converted to JSON, so only data types and structures that can be "JSONified" may be returned. Complex data structures like NumPy arrays should be converted to lists using the `my_array.tolist()` function. In addition, NumPy data types such as `numpy.float64` and others should be converted to the native Python equivalents. For example, convert a NumPy float to a Python native type with `float(my_numpy_float)`.

The screenshot shows the 'DEPLOY' tab of the DataScience Platform interface. The page is for a project named 'Space Exploration Jobs' and a specific job 'Iris Predict'. The interface includes a sidebar with navigation icons and a main content area with deployment settings.

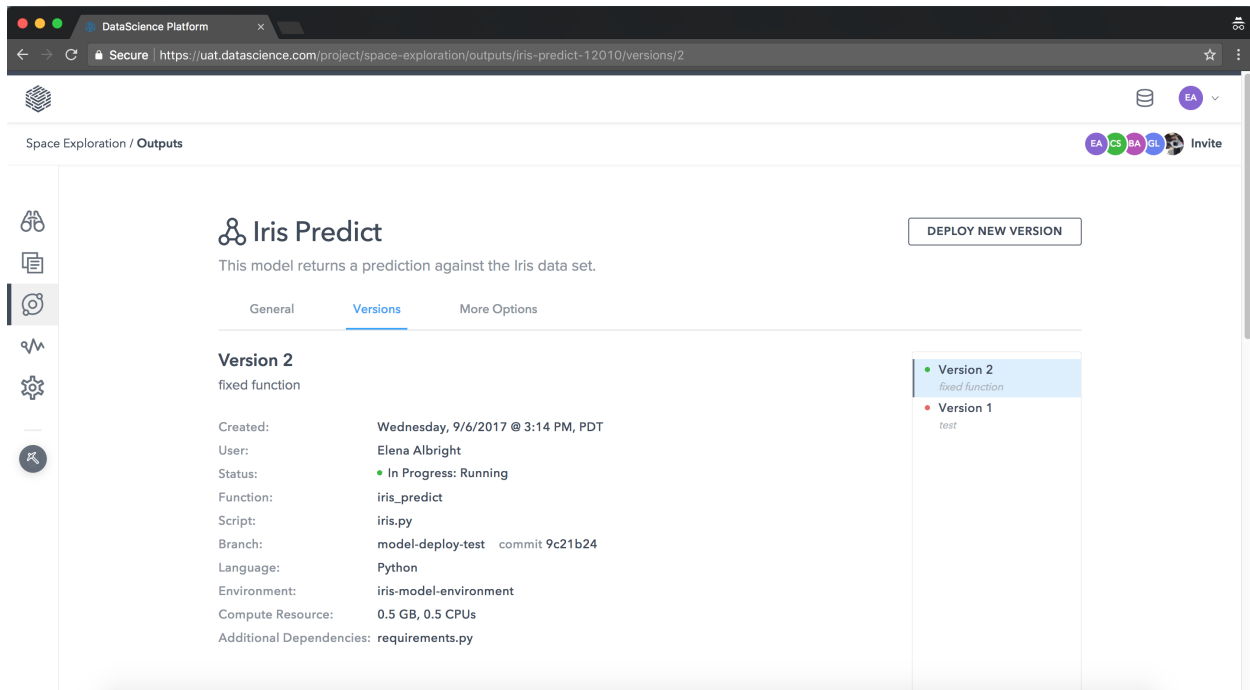
**Deployment Settings:**

- Run Every 15 Mins:** Set to 'No' (Runs every 15 minutes). A 'CHANGE' button is available.
- Branch:** master
- Language:** Python (commit: c981da9)
- Dependency Collection:** Standard
- Path to Model (Required, file must be in root directory):** A text input field with the placeholder 'Start typing...'.
- Select a Language:** A dropdown menu currently showing 'Python'.
- Compute Resource:** A dropdown menu showing '0.5 GB / 0.5 CPU'.
- Select an Environment:** A dropdown menu showing 'Default User Environment'.
- + Add Requirements (Optional):** A button to add additional dependencies.
- Specify Function:** A text input field.
- Give Example Data:** A button labeled 'Add example JSON data...'.

**Completed Runs:** A table with columns for 'Start time', 'Compute Resource', 'Select an Environment', and 'Status'. It currently shows 'No past runs found'.

### 10.1.2 Call an API

Once you have deployed an API, you can see information about it on its API page (in the project Outputs tab). On this page, you can run a sample call or find snippets for testing the API endpoint from your own code.



Copy/paste one of the cURL, node.js, or Python snippets your own code or command line to call the model.

In the code snippets, you'll find an API access token, which acts as a limited-access password to your endpoint. Share this token only with those you wish to call your API.

### 10.1.3 Manage an API

From the API page you can undeploy API versions. This action will shut down the container running the model. You can re-deploy later, which will re-create the container.

You may delete an API version, which shuts down the container and deletes the API image from the Platform's database. You may not undo a delete, but you may redeploy the same code after deleting a version.

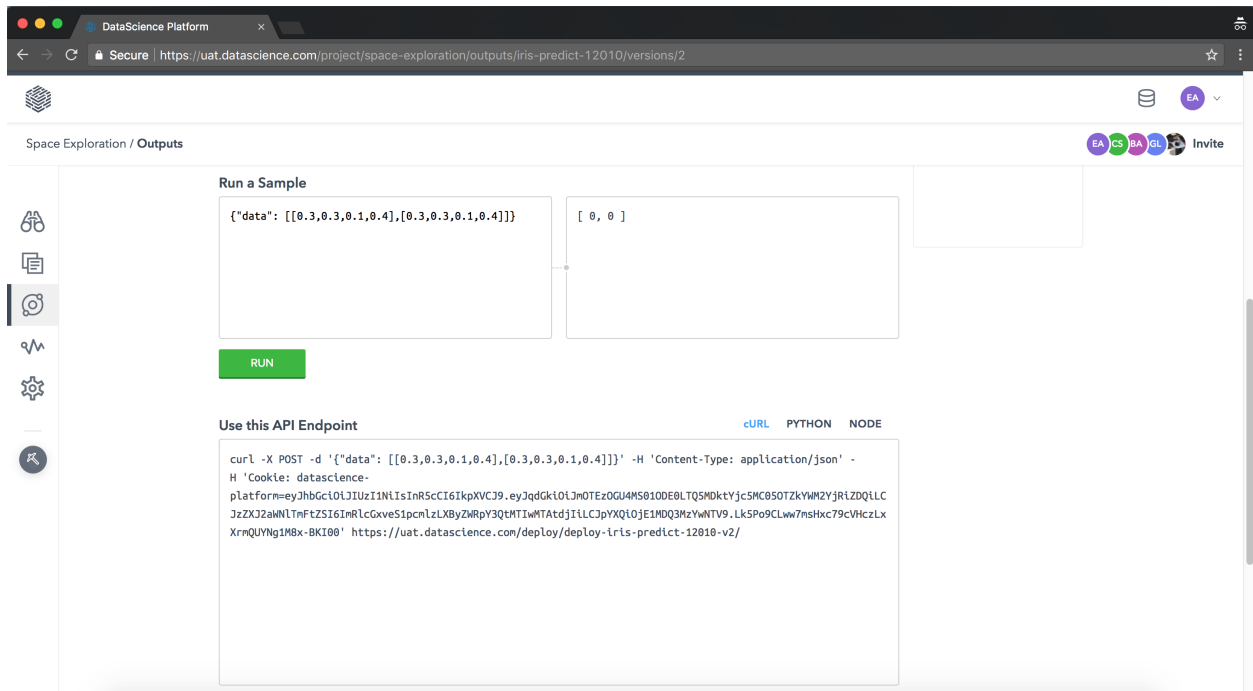
To delete all versions of a Model API, visit the "More Options" tab on the API page.

## 10.2 Best Practices: Deploying an API

Deploying a function or model as an API can help encapsulate data science work and expose it to other team members and applications. APIs can be used to do the following:

- Provide predictions as a microservice to a larger application.
- Provide inputs to visualization software like Tableau or Bokeh.
- Serve as a means of sharing your model with other analysts and data scientists.

This guide provides best practices for deploying APIs on the DataScience.com Platform.



### 10.2.1 Building the API Script

### 10.2.1.1 The Deploy Timeout

When you a deploy an API in the Platform, the following actions occur:

- A container is provisioned with the set specifications.
- The source files are included on the container.
- All dependencies indicated in requirements files will be installed via `pip`, `install.packages`, and `apt-get`.
- A webserver will be launched on the container.
- A number of workers or processes will be launched.
- Each of the workers will execute your script.

The web server uses a parameter called a timeout, which is set to 30 seconds. Any worker that is unresponsive for a period longer than the timeout is shut down. Thus, if the code in the API function takes longer than the timeout, the API will fail. Currently, the timeout cannot be configured. Before deploying your API, you can try running the script in a Platform Session to determine whether your code takes too long to execute. We recommend that the API function be as fast and lightweight as possible to minimize API response times.

### 10.2.1.2 Pickling vs. Training

You can include a model in your deploy script either by training a model in the script or by loading a serialized, pre-trained model. We generally recommend that you load serialized models in the model script rather than train a model. Loading a serialized model will result in faster build times and help avoid timeout issues.

For introductions to serializing data (pickling), see these articles written for [Python](#) and [R](#).

### Training a model within a deploy script:

In Python:

```
model = RandomForestClassifier()
model.fit(X, y)

def my_predict(data):
    return {'predictions': model.predict(data)}
```

In R:

```
model <- lm('y ~ x', df)

my_predict <- function(data){
  return(model, data)
}
```

Loading a serialized model within a deploy script:

In Python:

```
import pickle

with open(model_path) as model_file:
    model = pickle.load(model_file)

def predict(data):
    return {'predictions': model.predict(data)}
```

In R:

```
model <- readRDS(model_path)

my_predict <- function(data){
  return(model, data)
}
```

Pickled models can either be stored as part of the project repository, and therefore tied to version control, or can be stored in a remote file storage system like Amazon S3.

Some users prefer to keep the training process in the deploy script to promote reproducibility and transparency. The tradeoff to this is slower deployment.

### 10.2.1.3 Choosing a Response Type for a Deployed API

Once your model is deployed, every result it returns is converted to JSON before being sent over HTTP to the client. Anything the model returns must be a data structure that can be converted to JSON. Responses should be limited to combinations of strings, floats, integers, lists, or dictionaries.

For examples of how to transform common data structures into acceptable formats, see the table below. The leftmost column represents a given data structure in a particular language. The middle column represents how to transform the data structure to be JSON-friendly. “N/A” denotes that there is not an obvious means of conversion. “OK” denotes the data structure does not need to be transformed.

Original Structure	JSON-Friendly Transformation	Language
dict	OK	Python
numpy.ndarray	x.tolist()	Python
list	OK	Python
str	OK	Python
pandas.Series	x.to_dict()	Python
pandas.DataFrame	x.to_dict()	Python
list	as.matrix(as.data.frame(x))	R
data.frame	as.matrix(x)	R
character	OK	R
double	OK	R
matrix	OK	R

## 10.2.2 Deploying the API

### 10.2.2.1 Document Your Model or Function with a README

Documentation can help ensure that stakeholders and model consumers understand your model. Some things you may want to record include:

- Request signatures: Provide examples of what types of requests are valid. Describe the potential source(s) of the requested data.
- Response signatures: Provide examples of what the API returns so consumers know how to integrate results into their applications.
- A description of the model: As you and your organization accumulate more APIs, it will become difficult to recall the mechanics and details of every API. Including comments and metadata about training data, algorithms, hyperparameters, training time, etc. will help expedite onboarding, enable other team members to understand previous work, and make it easier to diagnose and improve the model in the future.
- A latency estimate: If other team members are going to be calling your model, it may be helpful to indicate how long your model takes to return responses. This will help ensure that model consumers can evaluate whether the model is fast enough for their application.
- Release notes: Any time the model is updated, either by retraining it, changing the request/response signatures, or changing the source code/choice of algorithm(s), it is important to update the documentation with details of the change. This will help avoid unintended errors and make it easier to revert to earlier versions if a rollback is needed.

### 10.2.3 Submitting Requests to Your API

To submit a request to your API, you'll need the API URL and its cookie string. You can find this data in the Versions tab of your model, under the Use this API Endpoint header, as shown below:

Below are examples of how to call your model API from R and Python.

Pass in a URL, a request body, specify the encoding, ignore SSL certs, and provide a cookie. Dictionaries are great data types to use for the body as part of the request. The keys of the dictionary will be taken as the names of the arguments of the deployed function.

In Python:

## Versions

More Options

## Version 12

added a verified number

### Run a Sample

[cURL](#) [PYTHON](#) [NODE](#)

```
curl -X POST -d '{"connection":{"src_bytes": 1, "src_dst_ratio": 1, "logged_in": 1, "dst_bytes": 1, "same_srv_rate": 1, "is_flow_flag": 1, "source_rate": 1, "error_rate": 1, "srv_error_rate": 1, "is_service_FTP": 1, "src_count": 1, "count": 1, "srv_error_rate": 1}}' -H 'Content-Type: application/json' -H 'Cookie: datacenterId=
```

```
import requests

url = 'https://myenv.datascience.com/deploy/mymodel/'
cookies = {
    'datascience-platform': 'my-cookie'
}
body = {
    'data': data.tolist()
}
predict = requests.post(url,
                        cookies=cookies,
                        verify=False,
                        json=body)
```

In R:

```
library('httr')
body <- list(data=mydata)
request <- POST(url,
               body = body,
               encode = 'json',
               config = config(ssl_verifypeer = 0L),
               set_cookies("datascience-platform" = cookie_string)
            )
```

### 10.2.3.1 Send More Records and Fewer Requests

If you are deploying an API to score requests, you can decrease the overall latency by sending more individual records per request. Concretely:

```
requests.post(url, cookie=cookie, verify=False, json = [user_1, user2])
```

will typically outperform

```
requests.post(url, cookie=cookie, verify=False, json = [user_1])
requests.post(url, cookie=cookie, verify=False, json = [user_2])
```

This is subject to the memory limits of the container executing the deployed API.

### 10.2.3.2 Run APIs on Larger Containers to Improve Response Times

If your deployed API is resource-intensive and the container running the API is undersized, API latency will increase. Besides working on reducing the resource load of your API, try deploying it on a container with larger resources. See our article on [allocating resources for containers](#) for details.

### 10.2.3.3 Use Resource Pool over On-Demand for Faster Builds

If your organization's environment supports on-demand containers, note that building on-demand containers requires the additional step of provisioning resources, which can take several extra minutes. To avoid longer builds, run APIs on your shared resource pool. This is particularly useful when prototyping.



#### 10.2.3.4 Prototyping to Production: Developing Internal Standards

If your model or function is being consumed by other team members and/or applications, it's important to note the following best practices. Best practices for different teams and use cases will vary, but ideally everyone on a team follows a common set of guidelines. These may include:

- Coding style guides, comments, and design patterns:
  - To improve its readability, your code should be written clearly, be well-commented, and follow coherent design patterns. See [Google's R style guide](#) and [PEP 8](#) for examples.
- Documentation:
  - Documentation can help others understand the context of your work and provide instructions on how to use it. Undocumented projects or models may fall into disuse after the original developer has left the project.
- Continuous Integration and Unit Testing:
  - These processes can help ensure that your model or function is working as expected at all times for all team members and applications. If you update the source code of your function, having an automated test suite can help avoid unexpectedly breaking other applications.
- Profiling and Optimization:
  - If others are consuming your API, it's important to set an acceptable latency. You want to ensure that expectations around latency are aligned. If you fall below this level, using memory and/or execution time profilers can help identify bottlenecks so you can address them.

### 10.2.4 Dependencies

Whether deploying models in Python or in R, it is preferable to have installations captured in requirements files rather than installing them directly in the model script. Once the deploy workers are built, the Platform will determine that an unresponsive worker is dead after a period of time and will kill the worker. Installations, especially those made using source files from the internet, do not take a defined amount of time so timeouts can occur if installations are part of the worker runtime.

#### 10.2.4.1 Python Libraries

You can list any packages your API requires in a `requirements-py.txt` file, and reference this file in the Deploy Configuration page. These packages will be installed from PyPI as part of the build process once you deploy your model. For details on Python requirements files, see the [pip user guide](#).

#### 10.2.4.2 R Libraries

List any R dependencies in a `requirements-r.txt` file. These packages must exist within the CRAN repository. Note that package versions cannot be specified in R requirements files; when a package is specified, the latest stable release will be installed. If you need to specify a version, you'll need to install the package within the API script itself.

#### 10.2.4.3 APT

Sometimes, especially when using certain R libraries, you will need dependencies on the Debian box running your model. These packages can be installed via `apt-get install`. Apt dependencies can be listed in a `requirements-apt.txt` file.

## 10.2.5 Examples

Below are examples of scripts you could deploy (the model) and scripts you could use to call the deployed model (the client).

### 10.2.5.1 Model

In Python:

```
import numpy as np
from sklearn.linear_model import LinearRegression

dim = 3
N = 100
X = np.random.normal(0, 1, size = (N, dim))
beta = np.random.normal(0, 1, size = dim)
err = np.random.normal(0, 1, size = N)
y = np.dot(X, beta) + err
model = LinearRegression().fit(X, y)

def predict(data):
    """Function to be deployed"""
    return model.predict(data).tolist()
```

In R:

```
dim <- 3
N <- 1000
beta <- rnorm(dim, mean = 0, sd = 1)
err <- rnorm(N, mean = 0, sd = 1)
X <- as.matrix(replicate(dim, rnorm(N, mean = 0, sd = 1)))
y <- as.matrix(X %*% cbind(beta) + err)

colnames(X) <- c('x1', 'x2', 'x3')
colnames(y) <- c('prediction')

df <- data.frame(cbind(X, y))

model <- lm("y ~ x1 + x2 + x3", df)

predictor <- function(data){
    return(predict(model, data))
}
```

### 10.2.5.2 Client

In Python:

```
import requests
from functools import partial

cookies = { 'datascience-platform': 'my-cookie' }
url = 'https://myenv.datascience.com/deploy/mymodel/'
predict = partial(requests.post, url, cookies=cookies, verify=False)

def encode(data):
```

```
    return {'data':data.tolist()}\n\nx1 = np.random.normal(0, 1, size = (1, 3))\n\npredict(json=encode(x1))
```

In R:

```
library('httr')\nurl <- 'https://myenv.datascience.com/deploy/mymodel/'\ncookie_string <- 'my-cookie'\nset_cookies("datascience-platform" = cookie_string)\n\nencode <- function(data){\n  return(list(data=data))\n}\n\npredictor <- function(data){\n  response <- POST(url,\n                   body = encode(data),\n                   encode = 'json',\n                   config = config(ssl_verifypeer = 0L))\n  return(content(response))\n}\n\nx1 <- as.matrix(replicate(3, rnorm(1, mean = 0, sd = 1)))\n\npredictor(x1)
```



Within this appendix, there is additional information related to articles in the For Users section of these docs. These supplementary articles are intended to provide further insights and direction beyond basic User actions.

## 11.1 Dockerfile Basics and Best Practices

In this section, you will learn how to create custom Docker images for your team of data scientists.

In order to build Docker images that contain the tools and dependencies your team needs, you need to write instructions in a Dockerfile, which is a text file that contains all the commands (in order) that need to be run to build the desired image. If you are not familiar with Dockerfile, we recommend reading this [Docker tutorial](#).

Prior to beginning, please review the following best practices and warnings for building environments on the Platform with Dockerfiles.

### 11.1.1 Best Practices

- If you reference files in your instructions, use relative paths, not absolute paths.
- Conda and R don't play well together. If you intend to have R and Python cross dependencies, avoid using Conda. Instead, install Python, R, and their respective libraries via `pip`, `R` and `apt-get` commands.
- Make sure that your Python and R interpreters are in your `PATH`. The version that is in your `PATH` will be executed. If you have installed multiple versions of the Python interpreter (e.g. Python 2.7, Python 3.6), make sure you activate the right one in your `PATH`. The same goes for R.
- Generally, we recommend having separate environments for Python 2 and 3.
- Take advantage of image “inheritance”. Build base images that could be used for other Base or User environments. Avoid creating images with very intricate sets of dependencies by breaking them into smaller images. This will help with debugging.
- Read the build logs very carefully. Sometimes installation errors will occur, yet the image build could still be successful.

## 11.1.2 Dockerfile Basics

This section outlines the basics of writing Dockerfiles. For those who are already familiar with Dockerfile, you may skip this section and proceed to the next one.

Below is a description of all Dockerfile instructions currently supported for Base and User environments.

### 11.1.2.1 Dockerfile Supported Instructions

#### 11.1.2.1.1 RUN Command

The `RUN` command executes shell commands (`/bin/sh -c` by default on Linux systems). Here's one example installing the Python package `gensim` using the package manager `Conda`:

```
RUN conda install --yes -n python3 gensim
```

You can chain shell commands within `RUN` by adding `&&` between each command. For example:

```
RUN conda install --yes -n python3 gensim && conda install --yes -n python2 gensim
```

You can run a variety of commands mostly related to package managers (`pip`, `conda`, `apt`) or others like `wget` or `curl`.

**Warning:** If you are using the `Conda` package manager, avoid creating a `Conda` environment. Instead, update the `root` environment with whatever dependencies you want to install.

As an example to the above warning, run this: `RUN conda env update -n root --file environment.yml` and avoid this: `RUN conda env create -f environment.yml`

#### 11.1.2.1.2 SHELL Instruction

You can change the default shell using the `SHELL` command. This changes all subsequent `RUN` commands. Simply add the following in your Dockerfile:

```
SHELL ["/bin/bash", "-c"]
```

This is an example where the bourne shell (`bash`) is used instead of the default.

#### 11.1.2.1.3 COPY Instruction

The `COPY` instruction is implicit in the Upload TAR button. After selecting a local tarball file, the tarball is exploded and all files will implicitly `COPY` to the Docker image. You do not need manually run `COPY`. All these supporting files will be available for you to use in Dockerfile instructions. For example:

```
RUN conda env update -n root --file environment.yml
```

#### 11.1.2.1.4 ADD Instruction

The `ADD` instruction is similar to the `COPY` instruction. As with the `COPY` instruction, files will automatically `COPY` to the Docker image with the Upload TAR button.

#### 11.1.2.1.5 ENV Instruction

The ENV instruction allows you to set environment variables in the Docker image. For example:

```
ENV my_variable itsvalue
ENV my_variable="itsvalue"
```

#### 11.1.2.1.6 USER Instruction

This sets the user name or UID of the user that is executing the instructions. Limit yourself to either `root` or `jupyter`. For example:

```
USER root
# or :
USER jupyter
```

### 11.1.2.2 Instructions Not Allowed

#### 11.1.2.2.1 ARG Instruction

The ARG instruction is not allowed. The ARG command usually precedes the FROM command, though an ARG command is not absolutely necessary.

#### 11.1.2.2.2 FROM Instruction

The FROM instruction is not allowed. It is generated implicitly when you select a base image. Every Dockerfile starts with a FROM statement. You can include comments throughout your file using `#`. For example:

```
# this is a dockerfile
FROM base_image
```

In a Dockerfile, all commands are capitalized and arguments are lower case. Here, ‘base\_image’ is the name of the base image from which you want to build a new image.

#### 11.1.2.2.3 CMD Instruction

The CMD instruction is not allowed.

#### 11.1.2.2.4 ENTRYPOINT Instruction

We do not allow custom ENTRYPOINT instruction at the present time.

#### 11.1.2.2.5 EXPOSE Instruction

The EXPOSE instruction is not allowed for either Base or User environments.

#### 11.1.2.2.6 VOLUME Instruction

We do not allow `VOLUME` instruction at the present time.

### 11.1.3 Putting It All Together

This section contains a few examples of Dockerfiles you can use in your workflow. The example starts with a base image that installs the package manager Conda. This base image should be built from the default base image environment.

#### 11.1.3.1 Example 1: Building a Conda Python 2.7 environment with ML and Stats Dependencies

##### 11.1.3.1.1 A Conda Base Dockerfile:

```
RUN wget --quiet https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.  
↪ sh && \  
/bin/bash Miniconda3-latest-Linux-x86_64.sh -f -b -p /opt/conda && \  
rm Miniconda3-latest-Linux-x86_64.sh  
ENV PATH /opt/conda/bin:$PATH
```

Note that Conda is located in `/opt/conda/bin`. It is added to the `PATH` in line 4. Also note that Miniconda3 is installed. That implies that Python 3.6 is installed. To install Python 2.7, see the user environment below. Alternatively, you can install Miniconda2.

##### 11.1.3.1.2 Example User Environment Dockerfile:

This is an example of a user environment Dockerfile where a user would select the Conda base environment, activate the Python 2.7 kernel, and install a series of packages with both Conda and pip package managers.

```
RUN conda install python=2.7 && conda install numpy && \  
conda install pandas && conda install scipy && pip install scikit-learn
```

Note that you don't specify version numbers for the libraries listed above. The most recent versions will be installed.

The diagram below shows the inheritance structure of the different Docker images for this example:

In the User environment, do not forget to select the tools you want.

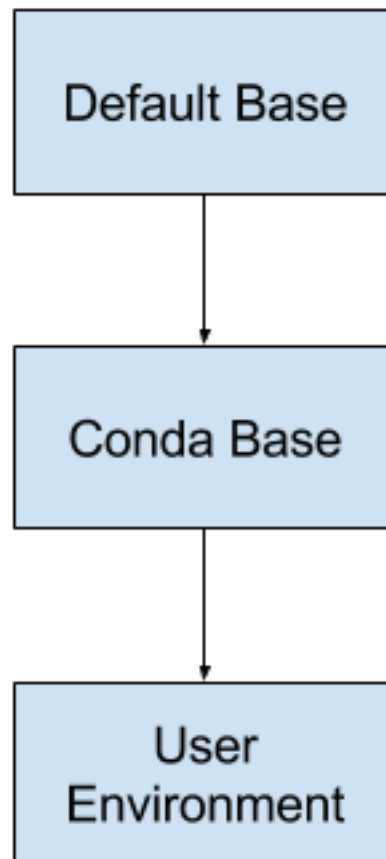
#### 11.1.3.2 Example 2: Installing R dependencies (rJava)

In this example, you only need to create a base image that contains all the dependencies needed to install `rJava`. In fact, you install `rJava` as part of the last command of this Dockerfile.

##### 11.1.3.2.1 A Base Dockerfile for rJava Dependencies:

```
USER root  
  
RUN apt-get update && \  
apt-get -y install default-jre && \  
apt-get -y install default-jdk && \  
pip install rJava
```





```
apt-get -y install r-base && \
apt-get -y install r-base-dev

RUN R CMD javareconf
RUN apt-get -y install r-cran-rjava
```

## 11.2 Enabling Hadoop and Spark

### 11.2.1 Introduction

The DataScience.com Platform provides seamless integration for Apache Hadoop, Hive, and Spark. The Platform will connect to your data where it lives, so there is no need to move data or add/replace costly infrastructure. To enable Hadoop, Hive, and Spark on your instance, you will need to follow a two-step process: (i) configure your Hadoop cluster and (ii) build your Hadoop-enabled environments.

### 11.2.2 Hadoop Cluster Configuration

To configure your cluster, navigate to the Hadoop Cluster tab in Admin Settings via the Avatar dropdown menu. Select your Hadoop provider from the dropdown list to begin.

The screenshot shows the 'Admin Settings' page with the 'Hadoop Cluster' tab selected. The page is titled 'Configure your external Hadoop Cluster'. It features a 'Choose a Provider:' dropdown menu with 'MapR' selected. Below this, there are fields for 'Cluster Name', 'CLDB IP', 'Version' (set to 5.2), 'Security' (set to MapR Ticketing), 'MEP' (set to 3.0), and 'SSL Truststore' (with an 'Upload' button). At the bottom, there are three checkboxes: 'Enable Hadoop', 'Enable Hive', and 'Enable Spark', all of which are currently unchecked.

Fill in the form with the cluster name, IP address, provider version, and additional security information. Then, choose to enable Hadoop, Hive, and/or Spark by checking the boxes. When you choose to enable a certain framework, you will be prompted to add additional information in the form of configuration files that you will upload into the form. In this next section, you will learn how to locate and acquire these files.

**Warning:** Please gather configuration materials from edge nodes or client machines that have successfully connected to the cluster. Cluster server notes may not have the proper configurations.

### 11.2.2.1 Enabling Hadoop (Optional)

#### 11.2.2.1.1 Optional Files

`core-site.xml`, `hdfs-site.xml`, `yarn-site.xml`, `mapred-site.xml`, `hadoop-env.sh`

These files can be found in `HADOOP_CONF_DIR`. It is usually symbolically linked to `/etc/hadoop/conf` but different distributions of Hadoop will lay them down in different places. Another common place for the actual files will be under the installation directory at `HADOOP_HOME/etc/hadoop`. If the files are empty or just the files with a `.template` suffix are present, it is not necessary to upload the files.

For more information, see Apache Hadoop's documentation on [Cluster Setup](#).

---

**Note:** Please note that some of the configuration options in portions of this documentation referring to services like HDFS NameNode, YARN NodeManager, etc. will not affect the operation of the cluster, since these services will not run inside a DataScience.com Hadoop-enabled environment. Instead, they will remain on the external Hadoop cluster that the environment will connect to.

---

As of yet, we don't override any Hadoop settings in the files that you upload, but we may override the settings in the files you upload for other services to ensure your clients can connect from inside the DataScience.com Hadoop-enabled environment. These settings will be outlined below.

### 11.2.2.2 Enabling Hive (Optional)

#### 11.2.2.2.1 Required Files

`hive-site.xml`

#### 11.2.2.2.2 Optional Files

`hive-env.sh`

These files can be found in `HIVE_CONF_DIR`. It is usually symbolically linked to `/etc/hive/conf`, but different distributions of Hadoop will lay them down in different places. Another common place for the actual files will be under the installation directory at `HIVE_HOME/conf`.

For more information, see Apache Hive's documentation on [Configuring Hive](#).

---

**Note:** Please note that `hive-site.xml` may have the Hive Metastore password present if you're connecting to Hive without any additional security authentication mechanism in place or if you take it from the server hosting `HiveServer(2)`.

---

We may override the following properties in `hive-site.xml`:

`hive.execution.engine`

`hive.metastore.schema.validation`

```
hive.metastore.sasl.enabled
hive.exec.scratchdir
```

#### 11.2.2.2.3 Tez

If Tez is enabled, we don't currently support uploading Tez-specific configuration files. Instead, at runtime we inject the appropriate configuration and properties to make sure the Tez jars are available on the `HADOOP_CLASSPATH` set in `hadoop-env.sh`, and that the `hive.execution.engine` is set properly in `hive-site.xml`.

### 11.2.2.3 Enabling Spark

#### 11.2.2.3.1 Required Files

```
spark-defaults.conf, spark-env.sh
```

These files can be found in `$SPARK_HOME/conf`. It is usually symbolically linked to `/etc/spark/conf`, but different distributions of Hadoop or an end user installation of Spark may or may not have set this up. The below files may not be actually be present in the configuration directory and only files with the `.template` suffix exist. In this case, it is not necessary to upload them to the Environments UI.

For more information, see Apache Spark's documentation on [Spark Properties](#) and [Environment Variables](#).

We override the following values in `spark-defaults.conf`:

```
spark.driver.extraJavaOptions
spark.executor.extraJavaOptions
spark.blockManager.port
spark.driver.port
spark.driver.blockManager.port
spark.driver.bindAddress
spark.driver.host
spark.sql.warehouse.dir
```

**Warning:** Since Hive and Spark are on different development cycles, when Spark integrates with Hive on a Hadoop cluster, it commonly uses a different configuration and is even packaged with a different version of Hive than is installed on the cluster. If that is the case, it is necessary to upload a `hive-site.xml` specifically for Spark. This should be found in `$SPARK_HOME/conf` as well.

## 11.2.3 MapR

### 11.2.3.1 Build a MapR Environment

Building a MapR environment is similar to building any Base environment, except all of the Dockerfile commands for installation are form-field driven. Simply navigate to the Environments screen, select Add Environment > Base Environment, and choose the Install Hadoop Dependencies option. Once selected, choose MapR as your provider, select the version and MEP, and build.

For more information about building environments, see our [Environment Management documentation](#).

Environment Name:  
My MapR Environment

Environment Description:  
This is an environment with Spark, Hadoop, and Hive enabled, connected to our MapR cluster.

Upload Readme

☐ Custom Dockerfile ☒ Install Hadoop Dependencies

From:  
Default Base Environment

Choose a Provider:  
MapR

Version:  
5.2.1

MEP:  
3.0

Frameworks:  
☒ Hive  
☒ Tez  
☒ Spark

NEXT STEP: BUILD →

After you have created an available MapR Base environment, create a User environment from this new Base environment to enable your users to connect to the cluster.

### 11.2.3.2 MapR Ticketing

The DataScience.com Platform supports the use of user-level MapR Tickets to authenticate against your MapR cluster. There is full support for MapR Ticket authentication across the Platform including interactive sessions, ad hoc and scheduled runs, and deployed APIs. To enable MapR Ticket authentication, select it as the Security option in the Cluster Configuration setup. Any users who want to authenticate to the cluster will need to upload their personal MapR Ticket to the Platform. See the [Account Setup documentation](#) for more information.

### 11.2.4 Other Providers

Don't see your provider? Contact [success@datascience.com](mailto:success@datascience.com).

## 11.3 Git Provider Integration

### 11.3.1 Introduction

Connecting projects to your code repository allows users to easily interact with branches and files from inside the Platform.

### 11.3.2 Supported Providers

- GitHub.com
- GitHub Enterprise 2.9+
- Bitbucket.org
- GitLab.com
- GitLab Enterprise 7+

### 11.3.3 GitHub OAuth Integration

Connecting the DataScience.com Platform to your GitHub repositories first requires an authentication integration. The following steps show how to create a GitHub app and connect it to the DataScience.com Platform.

#### 11.3.3.1 Create a GitHub OAuth Application

1. In the Settings tab of your GitHub organization, select OAuth Applications from the menu on the left.
2. Click the Register a new application button at the top right.
3. Fill out the form with the hostname that you used when you installed the Platform. The callback will be the same hostname, with the path `/oauth`. When you are finished, click Register application.
4. After you click Register application, make a note of the Client ID and Client Secret, as you will need these later in the integration process.

If you'd like to include an image for the GitHub OAuth Application, you may use [this image](#).

#### 11.3.3.2 Connect to Your GitHub OAuth Application

1. Inside the DataScience.com Platform, navigate to Admin Settings using the dropdown menu in the upper right-hand corner.
2. On the Admin Settings screen, click the Git Providers tab.
3. Choose your provider type from the dropdown (either GitHub.com or GitHub Enterprise). Provide the Client ID and Client Secret you obtained when creating the OAuth application.

##### GitHub Enterprise

---

**Note:** When integrating with GitHub Enterprise, you will also be prompted for the URL and the API URL. The URL will be the GitHub Enterprise instance you're connecting to, and the API URL will be the same URL appended with `/api/v3`. Example: URL: <http://github.datascience.com> API URL: <http://github.datascience.com/api/v3>

---

##### GitHub URL

**Warning:**

Do not append your organization name to the GitHub URLs in this section.

4. To connect your Platform account to GitHub, first navigate to your account's Settings page using the dropdown menu in the upper right-hand corner.

5. Click the Git Integrations tab, then click Add Credentials.
6. In the pop-up that appears, start typing the name of the Git provider you want to grant permissions to, then select it.
7. Click Connect, and you will be taken to GitHub for authorization. Click Authorize application to continue and you will be redirected back to the DataScience.com Platform.

### 11.3.4 Bitbucket Integration

The Bitbucket integration uses Bitbucket's App Passwords feature to grant the DataScience.com Platform access to your repositories. A Bitbucket App Password is just like your account password, but meant for other apps to control Bitbucket on your behalf.

Each user must bring their own Bitbucket App Password over to the DataScience.com Platform.

To enable the Bitbucket integration, visit Settings > Git Providers, select your Bitbucket type from the list of provider options, and fill in the details about your Bitbucket account.

### 11.3.5 GitLab Integration

How users authenticate with GitLab depends on the version of GitLab they'll be connecting to. GitLab 7 and 8 use passwords for authentication, while GitLab 9 and GitLab.com use access tokens.

Each user must bring their own GitLab access token over to the DataScience.com Platform. For end-user documentation on how to register a GitLab access token, see the Git Configuration section of our [user documentation](#).

To enable the GitLab integration as an administrator, visit Settings > Git Providers. Select GitLab from the list of provider options, then fill in the details about your GitLab account.

### 11.3.6 Manually Editing Providers in Postgres

1. To manually add providers in Postgres, first connect to the database endpoint using administrative credentials. We recommend using `psql` for this.
2. Once you have authenticated to the endpoint, ensure that you're connected to the 'platform' database. Then, delete the provider you intend to modify or remove. To create or recreate a provider, return to the DataScience.com Platform settings page and go to the Git Providers tab.

Make sure to replace the following placeholder values with your own: {git\_provider\_id}

```
SELECT * FROM git_providers; - find the id of the provider
DELETE FROM git_providers WHERE id = {git_provider_id};
```

3. To verify that your providers have been modified correctly, run the following query: `SELECT * from git_providers;`





### 12.1 Version 4.2.2 - October 4, 2017

The following release is a minor feature update, ready for installation on the *available* release channel.

#### 12.1.1 Features

##### 12.1.1.1 Select files to sync

- *This feature is previewed in this release.* From inside a Jupyter session, a user can select a subset of their modified files to sync to their project's repository.

##### 12.1.1.2 Resource Management for Users

- Users will be informed that their compute resource size selection is not available to select due to current resource constraints on the cluster.

### 12.2 Version 4.1.1 - September 20, 2017

The following release is a minor feature update, available on the *available* release channel.

#### 12.2.1 Features

##### 12.2.1.1 Built-in MapR Hadoop support for Hive and Spark

- For customers with MapR distributions of Hadoop, the DataScience.com Platform provides an easy form-driven method of configuring the cluster connection and installing all of the necessary dependencies.

## 12.3 Version 4.0.1 - September 6, 2017

The following release is a major feature update, ready for installation on the *available* release channel.

### 12.3.1 Features

#### 12.3.1.1 Environment Management

- Admins can now create and distribute customized, pre-installed collections of dependencies and packages as Environments to users on the Platform.

#### 12.3.1.2 Sync and Shutdown from Jupyter sessions

- It is no longer necessary to switch tabs between your work and your Session Details page to sync or shutdown your session.

#### 12.3.1.3 File path autocomplete

- Autocomplete your file path when running a script, publishing a report or app, or deploying an API

#### 12.3.1.4 Enhanced Platform availability

- Improved Availability of the Platform application, Load Balancer, and Database

#### 12.3.1.5 Single Sign On with SAML 2.0

- Integrate with your SAML 2.0 provider for authentication

## 12.4 Version 3.9.1 - August 23, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.4.1 Features

#### 12.4.1.1 Report versioning

- Users can now create multiple versions of a report under the same URL.
- Users can also edit report version titles and descriptions.

#### 12.4.1.2 User-supplied custom tagging for Amazon EC2 on-demand resources

- End users can now input custom tagging to Amazon's EC2 metadata when provisioning on-demand resources.

## 12.5 Version 3.8.1 - August 9, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.5.1 Enhancements

- Support for Gitlab version 7+ with API version 3
- Support for Redhat Enterprise Linux version 7.3
- Avatars with profile images
- Minor bug fixes and security enhancements

## 12.6 Version 3.7.1 - July 26, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.6.1 Features

#### 12.6.1.1 R Shiny dashboards deployable to the Outputs page

- Users can now publish R Shiny applications to a dedicated Shiny server on the Platform, then share links to applications with project collaborators and business stakeholders.

#### 12.6.1.2 Resource Management Dashboard

- Admins can now manage all the server and Docker container resources running in the Platform: monitor RAM and CPU usage, identify unhealthy servers or analyses, and shut down unwanted processes.

## 12.7 Version 3.6.1 - July 13, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.7.1 Features

#### 12.7.1.1 H2O.ai dependency collection

- This dependency collection has H2O and its dependencies pre-installed to improve your AI capabilities

### 12.7.2 Enhancements

#### 12.7.2.1 Enhanced support for Internet Explorer 11

- Improved experience of operating the Platform on the latest version of Internet Explorer

### 12.7.2.2 Shortcut links for returning to interactive sessions in progress

- Navigate directly into your running Jupyter or RStudio session from the Currently Running dropdown

## 12.8 Version 3.5.1 - June 28, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.8.1 Features

#### 12.8.1.1 Administrator-configured compute resources sizes

- Administrators of the Platform can now configure the compute resource size options that are made available to users of the Platform from the Admin Console.

#### 12.8.1.2 Various user experience and usability enhancements

- Users can now name their sessions to distinguish them more easily from each other.
- On session shutdown, users will be alerted to any un-synced changes in the session to prevent unintentional loss of work.
- On a run details page for any past run, users will have the option to “rerun” the script but at the latest commit, enabling users to quickly iterate while maintaining environment configurations.

## 12.9 Version 3.4.1 - June 15, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.9.1 Features

#### 12.9.1.1 Curated dependency collections

- Get started quickly on difficult data science problems with new dependency collections. New dependency collections now include the Standard set of packages as well as curated powerful packages for solving specific data science problems.
  - RStudio: Time Series
  - Jupyter: Deep Learning, Bayesian Analysis, and NLP

#### 12.9.1.2 Multiple language kernels available in Jupyter sessions

- Python 2.7, Python 3.5, and R 3.2 are available in each session.

## 12.10 Version 3.3.1 - June 7, 2017

The following release is a minor feature update, available on the *Stable* release channel.

## 12.10.1 Features

### 12.10.1.1 GitHub Enterprise and GitLab Enterprise integrations

- In addition to GitHub.com, Bitbucket.org, and GitLab.com, the latest release also supports the enterprise versions of GitHub and GitLab for version control, allowing you to base your projects off of work in those repositories.

### 12.10.1.2 Global Environment Variables

- Platform Admins can now set environment variables at the global level so that secrets needed across projects can be set once and managed in one place. Each key-value pair can have user- and team-level permissions, ensuring control and security.

### 12.10.1.3 On-demand compute resources in AWS VPCs

- For installations in customers' Amazon VPCs, you can now control your cloud footprint and, therefore, your costs by provisioning single-use compute resources for sessions, runs, and deployed APIs. From the Platform, spin up an EC2 instance of your choosing, do your work, and shut it down when you no longer need that machine.

### 12.10.1.4 LDAP

- Admins can connect their company's active directory user management system to the DataScience.com Platform. LDAP-enabled environments use the specified active directory to authenticate users. Admins can manage Users in their referred central location.

## 12.11 Version 3.2.1 - May 31, 2017

The following release is a minor feature update, available on the *Stable* release channel.

### 12.11.1 Features

#### 12.11.1.1 Bitbucket.org and GitLab.com integrations

- In addition to Github, the latest release also supports Bitbucket.org and GitLab.com for version control, allowing you to base your projects off of work in those repositories.

#### 12.11.1.2 RStudio

- In addition to Jupyter, users can launch RStudio interactive sessions where they can import pre-installed packages, use environment variables, and sync directly to their repository.

#### 12.11.1.3 Publish RMarkdown HTML docs

- After creating analyses in RStudio, publish your findings for your teammates and colleagues as sharable, reproducible reports.

## 12.12 Version 3.1.1 - May 4, 2017

The following release is a major feature update. It is now available as version 3.1.1.

### 12.12.1 Features

#### 12.12.1.1 Projects

- Ensure visibility of your team's work by organizing code, data, and outputs into projects. Centralize knowledge, invite collaborators, and do work that is focused on solving the problems that impact your organization.

#### 12.12.1.2 GitHub Integration

- Version your shared code and watch your projects progress. Track decisions, milestones, and project lineage over time, and never lose sight of your work again.

#### 12.12.1.3 Secret Management

- Avoid committing secrets to code by storing them as secure environment variables.

#### 12.12.1.4 Launch Jupyter Interactive Sessions

- Quickly spin up a Jupyter interactive session with support for Python 2, Python 3, and R.

#### 12.12.1.5 Publish Reports

- Turn Jupyter notebooks, markdown documents, and other files from your project into reproducible reports that can be shared across your organization.

#### 12.12.1.6 Deploy APIs

- Deploy Python and R code behind a REST API to make it instantly available for integration with real-time applications or dashboards.

#### 12.12.1.7 Run Scripts

- Run Python and R scripts from a web UI, and share outputs across your team.

#### 12.12.1.8 Schedule Runs

- Run code on a schedule to automate data science processes.

---

## Tutorials and Examples

---

This section contains supplementary information to help you work efficiently on the DataScience.com Platform. Peruse our collection of learning modules, tutorials, and examples for in-depth guidance and additional support.

### 13.1 Learning Modules

The following learning modules provide extensive support for users wishing to complete specified actions within the DataScience.com Platform. Each module guides users through in-depth learning materials containing videos, screenshots, and written instructions. Navigate through the materials at your own pace and download the supplementary templates and files to facilitate your own work. You can find best practices and/or FAQ sections at the end of each module.

#### 13.1.1 Use Shiny on the DataScience.com Platform

**Using Shiny on the DataScience.com Platform:** In this lesson, you'll learn some of the basics of Shiny, including what it is and what you can do with it. Specifically, you will learn how Shiny works on the DataScience.com Platform, as well as the advantages to using the Platform to host your apps. Through the tutorials, you'll have the opportunity to build an example app based on a template we provide, and you'll likewise learn how to produce some simple UI variations to your app. Finally, you'll learn how to publish your app on the DataScience.com Platform in order to easily communicate your insights throughout your organization.

#### 13.1.2 Connect Tableau to Model APIs on the DataScience.com Platform

**Connect Tableau to the DataScience.com Platform:** In this lesson, you'll learn about another way you can use the tools you love with the DataScience.com Platform. We'll review everything you need to know to connect your Tableau software to a deployed model in the Platform in order to retrieve and visualize predictions. We will set up Tableau's Web Data Connector, which will route a request to our deployed model through Flask, an easy-to-set-up server. We'll also provide templates that correspond to our examples in order to enable you to set up your own connection with ease.

## 13.2 Examples

### 13.2.1 How to Create and Deploy a Shiny App

Shiny by RStudio is a framework for turning R code into interactive dashboards. You can build RShiny apps on the DataScience.com Platform within the familiar R or RStudio environment. RShiny app is a collection of .R scripts. This quick tutorial shows how to build an interactive map with RShiny using a dataset on Uber pickups in New York City. For a more extensive tutorial on this topic, visit the related Learning Module: [Use Shiny on the DataScience.com Platform](#)

RShiny apps consist of two main components:

- Code to be executed by the server that will power the app
- Code defining the UI appearance.

These two components can be defined in separate .R files or combined in one app.R file. In this tutorial, you'll see an example of both components defined inside one app.R script. This script can also contain any data manipulations needed.

#### 13.2.1.1 Loading the Data

First, import libraries and set your color theme:

```
library(tidyr)
library(ggplot2)
library(scales)
library(mapproj)

# Set plotting parameters and import color palettes
set_ds_theme = function() {
  theme_update(panel.background = element_rect(fill = "white", colour = "grey50")
    ,plot.title = element_text(hjust = 0.5))

  # Color palette for continuous data
  cont_ds_palette <- c('#f7fbff', '#d7ecfd', '#3ba3f8', '#328bd5', '#266aa2'
    , '#000000')
  cont_gradient <- colorRampPalette(cont_ds_palette)

  # Palette for discrete/categorical data
  cat_ds_palette <- c("#3ba3f8", "#afb6bd", "#3eb642", "#6981ef",
    "#b353b5", "#e66867")
}
set_ds_theme()
```

The data is stored on our public AWS S3 bucket. You can access this data with your own set of AWS keys. The data is pulled in and the average number of Uber pickups by location and hour is calculated:

```
bucketName = "ds-site-static-assets"

s3load('s3://ds-site-static-assets/ds-examples/rshiny/Uber/data/processed_uber_nyc.
  ↪RData',
      bucket = bucketName,
      key=Sys.getenv("AWS_ACCESS_KEY_ID"),
      secret=Sys.getenv("AWS_SECRET_ACCESS_KEY") )
```



```
# Precalculate mean pickups in each location by hour
locID_zone_dim <- unique(agg_data[, c("locationID", "zone")])
agg_data$hour <- as.numeric(agg_data$hour)
geo_pickups <- agg_data %>%
  group_by(locationID, hour) %>%
  summarize(mean_pickup = mean(pickups))
```

### 13.2.1.2 Defining the UI Components

Next, define the main UI components of the app. The goal is to display a heatmap of Uber pickups for NYC. To make the map interactive, you can add a slider filter for changing the hour and a tooltip displaying location coordinates. All of these components will be defined in the UI section:

```
# Define UI for application that draws a heatmap
ui <- fluidPage(
  # Application title
  titlePanel("Pickups by Hour")

  # Sidebar with a slider input for number of bins
  , sidebarLayout(
    sidebarPanel(
      sliderInput("hour",
                  "Hour of day:",
                  min = 0,
                  max = 23,
                  value = 24)
    )

    # Show a plot of the generated distribution
    , mainPanel(
      plotOutput("NYC_heatmap"
                 , height = 700
                 , width = 700
                 , hover = hoverOpts(id = "plot_hover"))
    )
  )
  # Tooltip
  , fluidRow(column(width = 6
                    , verbatimTextOutput("tooltip")))
)
```

### 13.2.1.3 Defining the Server Component

Finally, define the server component. This will be a function that takes in an input from UI components and returns a plotting output. Each output component corresponds to a chart or UI element. In this case, the app takes in the hour of the day from slider as input and returns a heatmap plot. Another output returned is coordinate values for map tooltip. Note that output names `NYC\_heatmap` and `tooltip` get referenced in the UI component.

Finally, tell the app where to find server and UI components. This part can be omitted if the UI and server get defined in two separate .R scripts.

```
# Run the application
shinyApp(ui = ui, server = server)
```

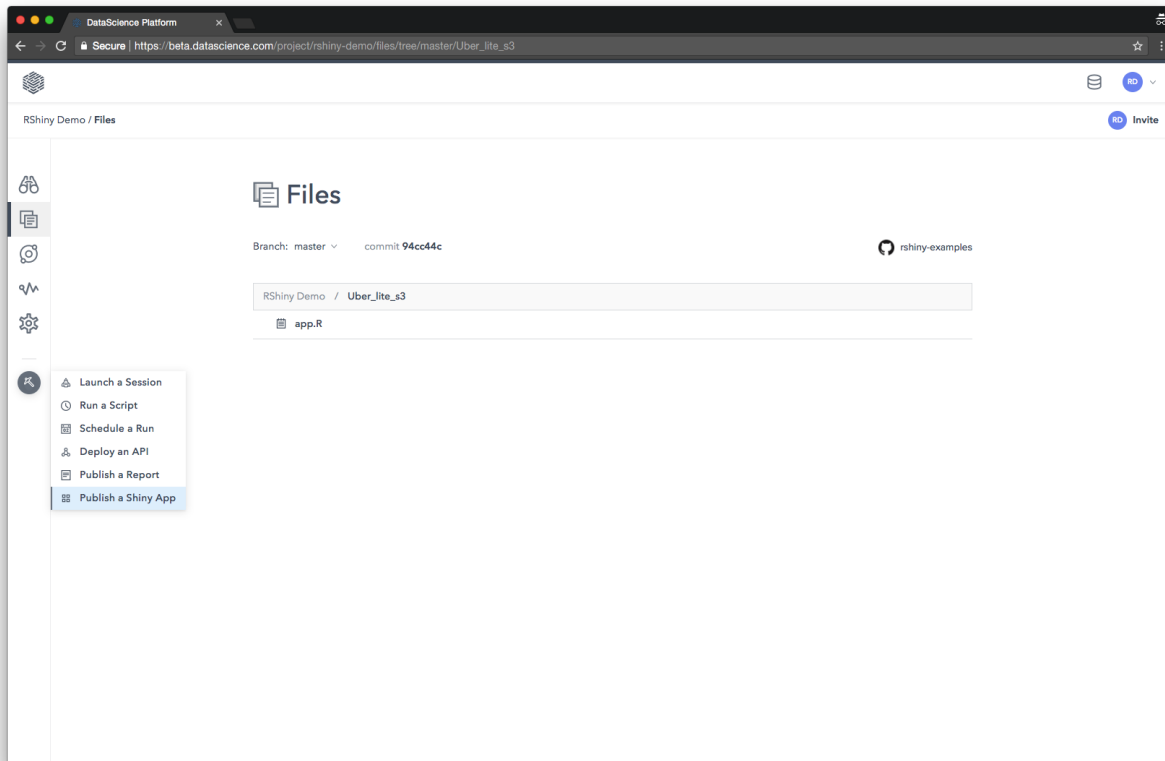
### 13.2.1.4 Running the App

All of the above snippets are combined in app.R script. To run the app, simply run this from same directory as app.R:

```
library(shiny)
runApp("app.R")
```

### 13.2.1.5 Publishing the App

Now the app is ready to be published as an Output. To publish, click on the Quick Actions button and select Publish a Shiny App:

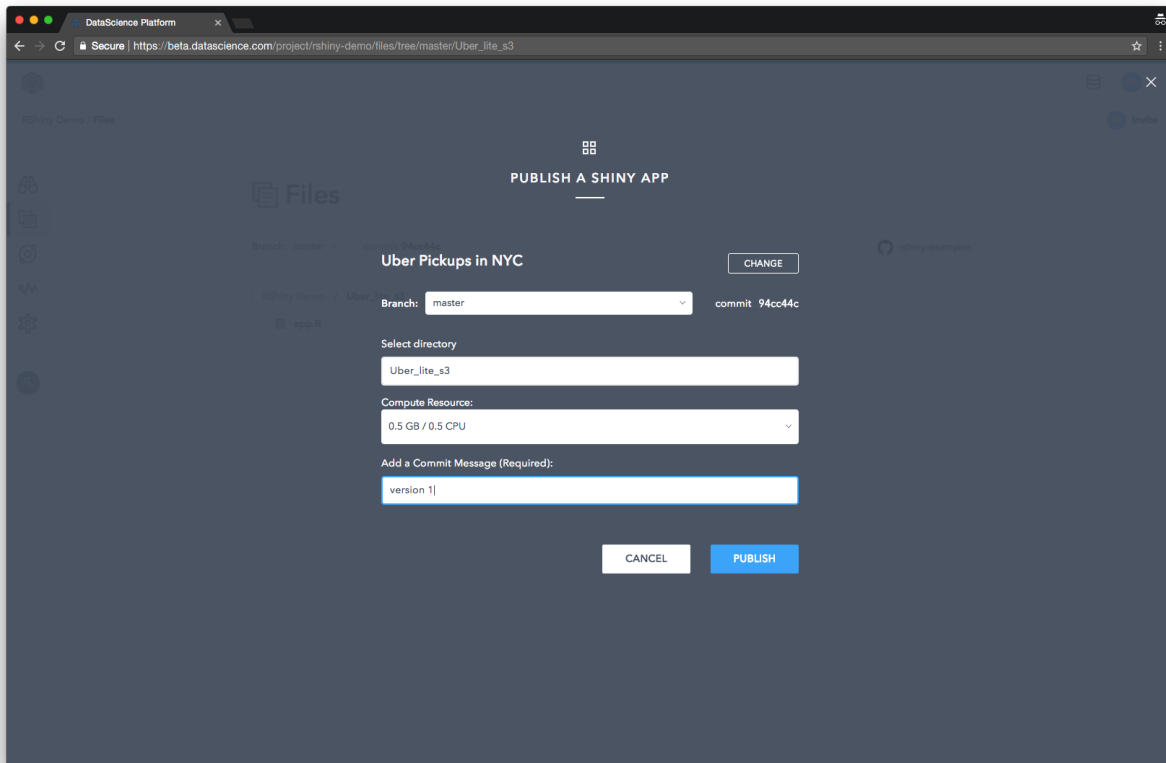
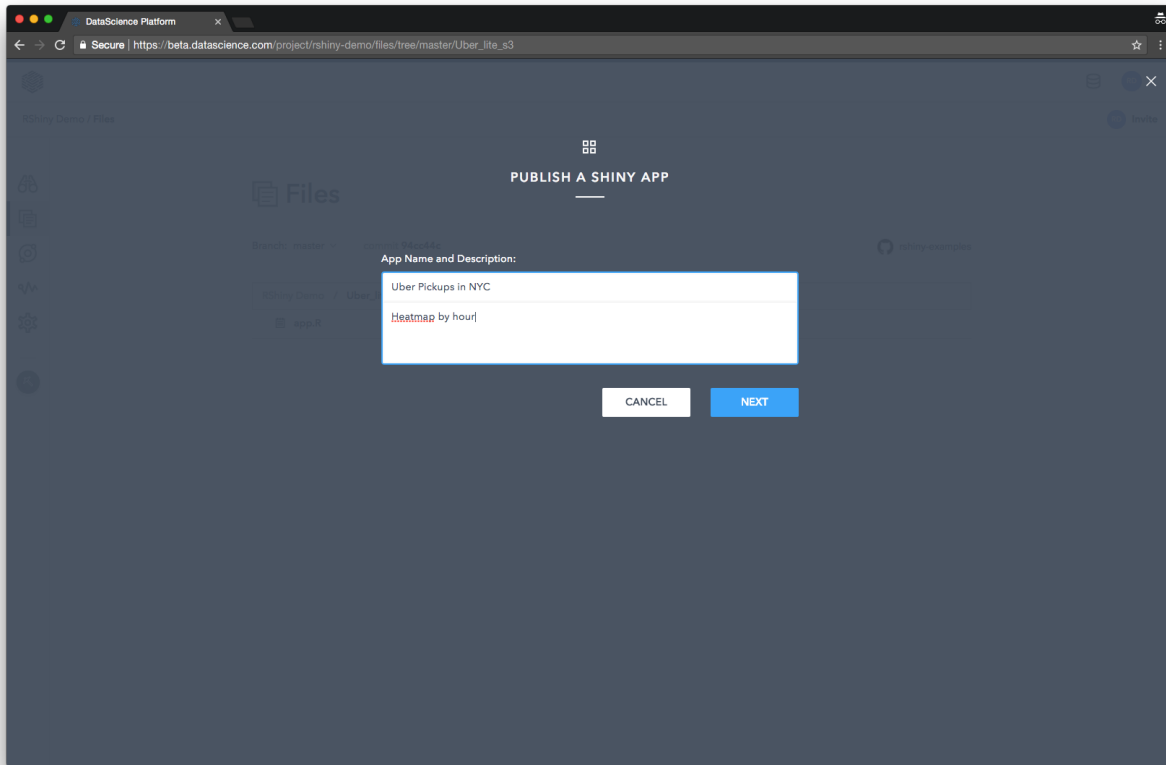


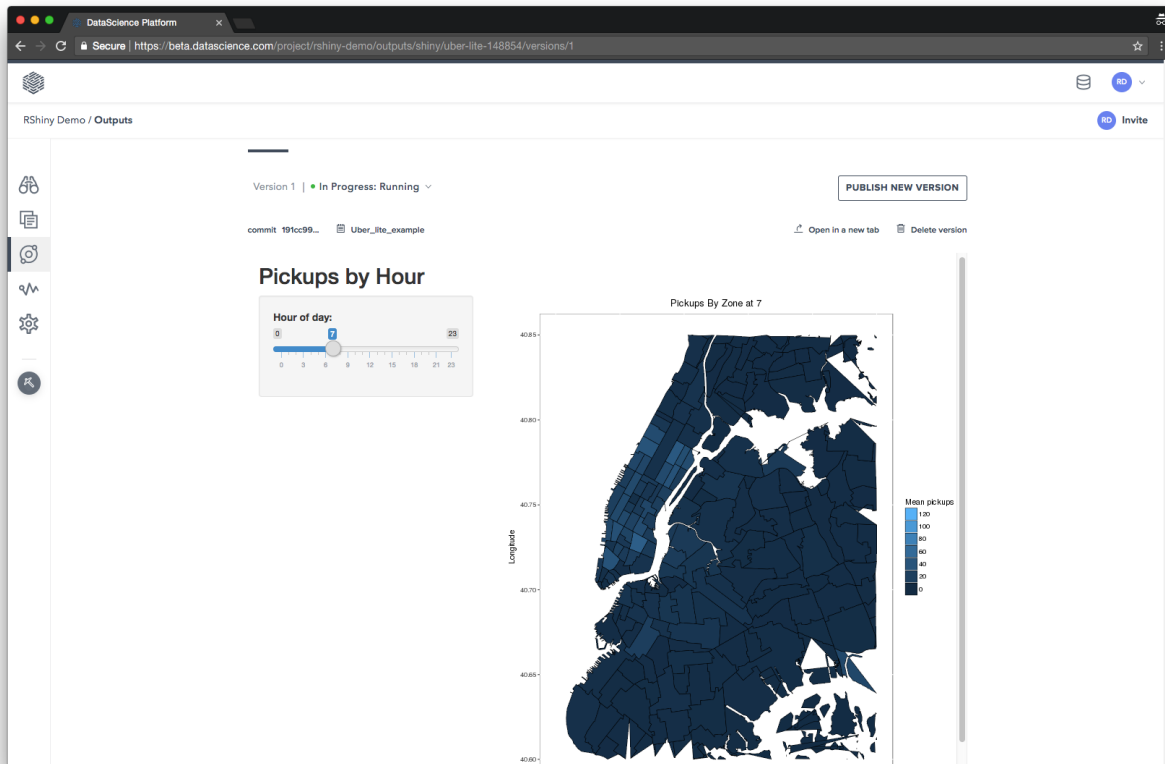
Give your app a name and a short description and click Next:

Select your repo branch, app directory, compute resource size, and provide a commit message. Then click Publish:

**Warning:** You must specify a directory, not a '.R' file when publishing a Shiny dashboard.

Now your app is visible under Outputs. Users can view and interact with the app without having to start a new R session:





## 13.2.2 Using a Deployed API

The purpose of this article is to show an example of a use case for a deployed API. An API can be used to automatically score incoming data with a pre-trained deployed model.

### 13.2.2.1 Business Use Case

This article considers a major hotel chain as an example. The hotel chain uses customer reviews to identify potential problems. The management wants to reduce time spent reading through large volumes of customer reviews. In this case, the deployed API will take reviews as input and assign category labels. Labeled reviews can then be sorted, summarized with a BI tool, and addressed by appropriate departments.

### 13.2.2.2 Training the Model

We will use a technique called Topic Modeling (Latent Dirichlet Allocation, or LDA) to discover topics mentioned in reviews. This algorithm will search through the review texts and summarize them in a handful of topics as words and phrases customers tend to use together. Training the model is done using the `gensim` Python library. First, read in the data:

```
# Platform Kernel: python2
# Libraries: boto==2.48.0, pandas==0.20.3, numpy==1.13.1, gensim==2.3.0, nltk==3.2.4,
↳ re==2.2.1, requests==2.18.3

import os
import pandas as pd
```

```

import numpy
import string
import re
from gensim import corpora
from gensim.models import Phrases
from gensim.models.ldamodel import LdaModel
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# utility functions for reading/writing to AWS s3
def get_bucket():
    conn = S3Connection(access_key, secret_key)
    return conn.get_bucket(env_name, validate=False)

def pull_file_from_s3(self, key, tmp_localdir=''):
    s3_bucket = self.get_bucket()
    payload = s3_bucket.get_key(key)
    if not os.path.exists(os.path.dirname(tmp_localdir+key)):
        os.makedirs(os.path.dirname(tmp_localdir+key))

    local_file = payload.get_contents_to_filename(tmp_localdir+key)
    print "Grabbed %s from S3. Local file %s is now available." % (key, tmp_
↪localdir+key)

def write_obj_to_s3(obj, localpath, key) :
    '''Write pickled object to s3. localpath should be the same as key.
    The file will be written to localpath first and then transfered to s3
    with key.

    '''
    if key != None :
        name=key
    else :
        print('Specify an s3 key')
    k=name.replace(' ','-')
    print('Modified s3 key %'.format(k))
    s3_key=Key(get_bucket())
    s3_key.key=k
    pickle.dump(obj,open(localpath,'wb'))
    s3_key.set_contents_from_file(open(localpath,'r'))
    print("Sent obj %s to S3 with key '%s'"%(localpath,k))
    def pull_pickle_from_s3(key, tmp_localdir=''):
        '''
        Grab pickled object from s3
        '''
        local_path = tmp_localdir+key
        local_dir = os.path.dirname(local_path)
        if not os.path.exists(os.path.dirname(tmp_localdir+key)):
            os.makedirs(os.path.dirname(tmp_localdir+key))

        s3_bucket = get_bucket()
        payload = s3_bucket.get_key(key)
        local_file = payload.get_contents_to_filename(local_path)
        print("Grabbed %s from S3. Local file %s is now available." % (key, key))
        return pickle.load(open(local_path, 'rb'))

```

```

dir_path = 'my-path-to-s3/hotel_reviews.txt'
pull_file_from_s3(dir_path, tmp_localdir=' ')

f = open('../'+dir_path, 'r')
raw=f.read()
f.close()

# process the text file
lines = raw.splitlines() # split on lines and carriages

reviews = [line.strip('<Content>') for line in lines if '<Content>' in line]

```

Next, apply standard cleaning by removing bad characters and stop words, and applying stemming. Finally, the gensim library is used to convert clean word tokens into a dictionary and bag-of-words corpus for modeling:

```

# Utility functions for text processing
def default_clean(text):
    '''
    Removes default bad characters
    '''
    if not (pd.isnull(text)):
        text = filter(lambda x: x in string.printable, text)
        bad_chars = set(["@", "+", "<br>", "<br />", "/", "\"", "'", "\\", '(', ')', '<p>
↪ ',
                                '\n', '<', '>', '?', '#', ',', '.', '[', ']', '%', '$', '&',
↪ ';',
                                '!', ':', '-', '*', '_', '=', '}', '{'])
        for char in bad_chars:
            text = text.replace(char, " ")
        text = re.sub('d+', "", text)

    return text

def stop_and_stem(text, stemmer = PorterStemmer()):
    '''
    Removes stopwords and does stemming
    '''
    stoplist = stopwords.words('english')

    text_stemmed = [[stemmer.stem(word) for word in document.lower().split()
                      if word not in stoplist] for document in text]
    return text_stemmed

clean_reviews = [default_clean(d).lower() for d in reviews]
stemmed = stop_and_stem(clean_reviews)

# clean up raw reviews and prepare dataset for model
numpy.random.seed(seed=44)
dictionary = corpora.Dictionary(stemmed)
corpus = [corpora_dict.doc2bow(t) for t in stemmed]

```

### 13.2.2.3 Fitting the Model

Now, fit the model with five topics. Note that usually the number of topics is chosen through optimizing some goodness of fit metric such as topic coherence or log-perplexity.

```
# number of topics
K=5

# Run LDA model to extract topics
lda = LdaModel(corpus=corpus, id2_word=dictionary, num_topics=K, passes=10)
```

### 13.2.2.4 Saving the Model

Trained models can be saved in a serialized format on AWS S3 to be accessed by the API later:

```
# Save the model
write_obj_to_s3(lda, 'lda_model', 'my-path-to-s3/lda_model')
write_obj_to_s3(dictionary, 'dictionary', 'my-path-to-s3/dictionary')
```

### 13.2.2.5 Deploying the Model

To deploy the trained model, write a function that takes in a review as input and produces a label or list of labels as output. This function can use the serialized model we just trained and saved. The deploy function and any supporting code should be saved in a .py script, which will get deployed behind an API. This deploy script is what will make predictions when new data hits the API:

```
def pull_pickle_from_s3(key, tmp_localdir=''):
    '''
    Grab pickled object from s3
    '''
    local_path = tmp_localdir+key
    local_dir = os.path.dirname(local_path)
    if not os.path.exists(os.path.dirname(tmp_localdir+key)):
        os.makedirs(os.path.dirname(tmp_localdir+key))

    s3_bucket = get_bucket()
    payload = s3_bucket.get_key(key)
    local_file = payload.get_contents_to_filename(local_path)
    print("Grabbed %s from S3. Local file %s is now available." % (key, key))
    return pickle.load(open(local_path, 'rb'))

def max_topic(scored_list):
    return max(scored_list, key=lambda item: item[1])[0]

# read in the model
dictionary = pull_pickle_from_s3('my-path-to-s3/dictionary')
lda_model = pull_pickle_from_s3('my-path-to-s3/lda_model')

# Main deploy function
def label_review(new_review):
    '''
    Take a new review as a list and assign it to one of the existing pre-trained_
    topics
```

```

'''

# pretty topic labels
name_dict = {0: "Front Desk",
              1: "Pool Feedback",
              2: "Restaurant and Bar Service",
              3: "Happy Customers",
              4: "Complaints"}

# transform text into the bag-of-words space
clean_review = [default_clean(d).lower() for d in new_review]
stemmed = stop_and_stem(clean_review)

# Predict label
new_vector = [dictionary.doc2bow(t) for t in stemmed]

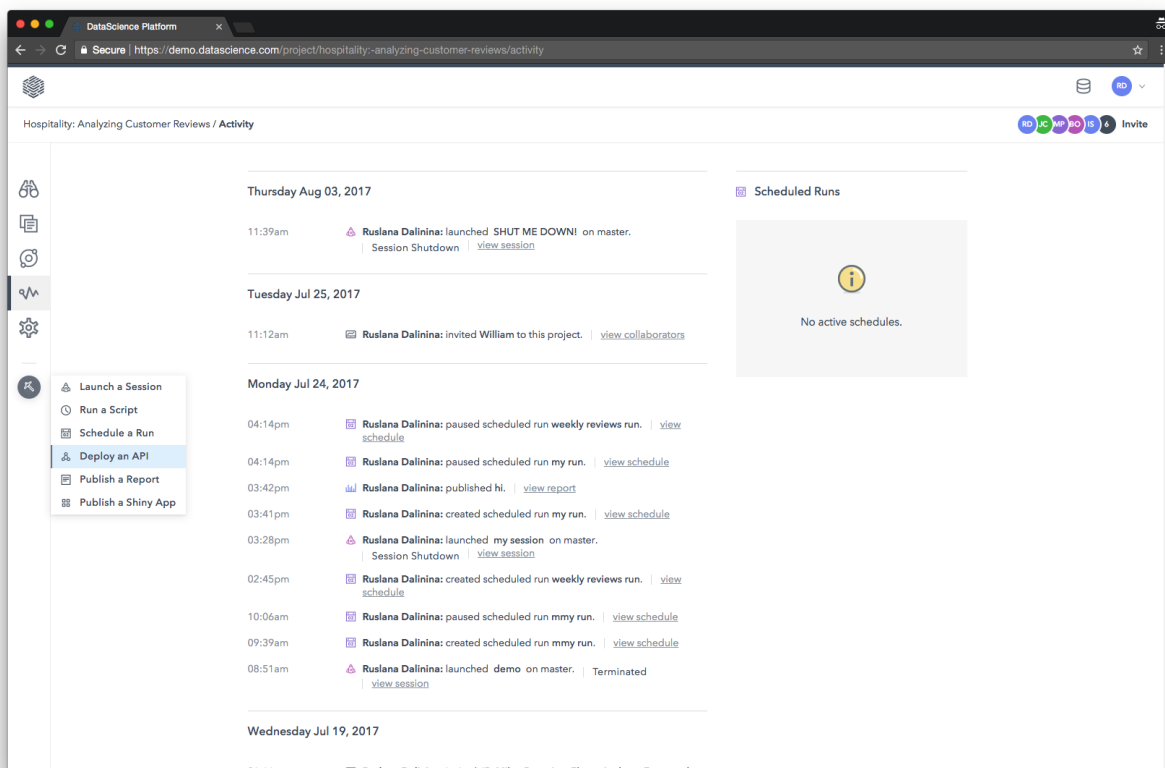
lda_vector = lda_model[new_vector]

id_ = map(max_topic, lda_vector)

print("Review Categories")
return map(name_dict.get, id_)

```

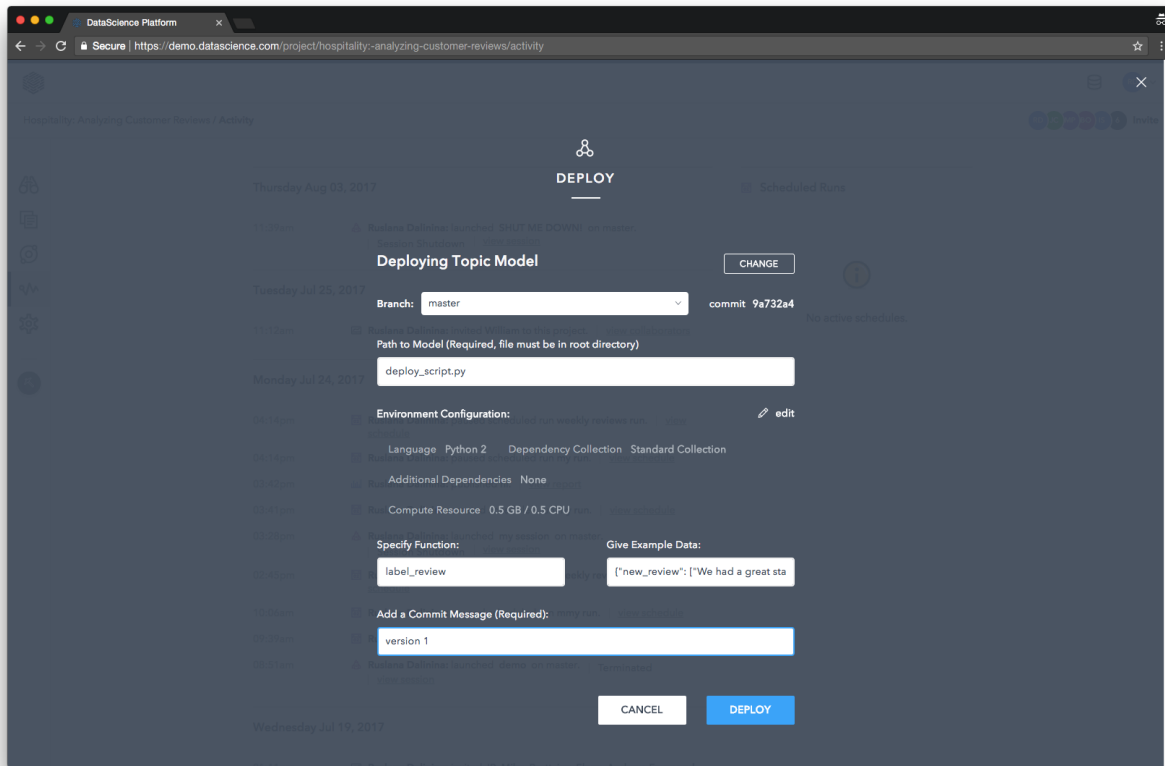
Now you are ready to deploy the model. Use the Deploy API option found under Quick Actions and go through the prompts:



Specify your branch, deploy script name, deploy function name, and example input:

The last step is to specify the dependencies of your deployed function. This is done by clicking the Edit option of





Environment Configuration. Simply include the name of your pip requirements file. `deploy\_requirements.txt` lists all package dependencies needed for the deploy function to run. It is recommended to put that file in the same folder as the deploy Python script.

The review topic model is now deployed behind an API. You can access all currently running APIs in the Outputs tab of the Platform.

### 13.2.2.6 Calling the API

The batch of new incoming reviews can now be passed as an input to the API. The text below shows an example of a raw review.

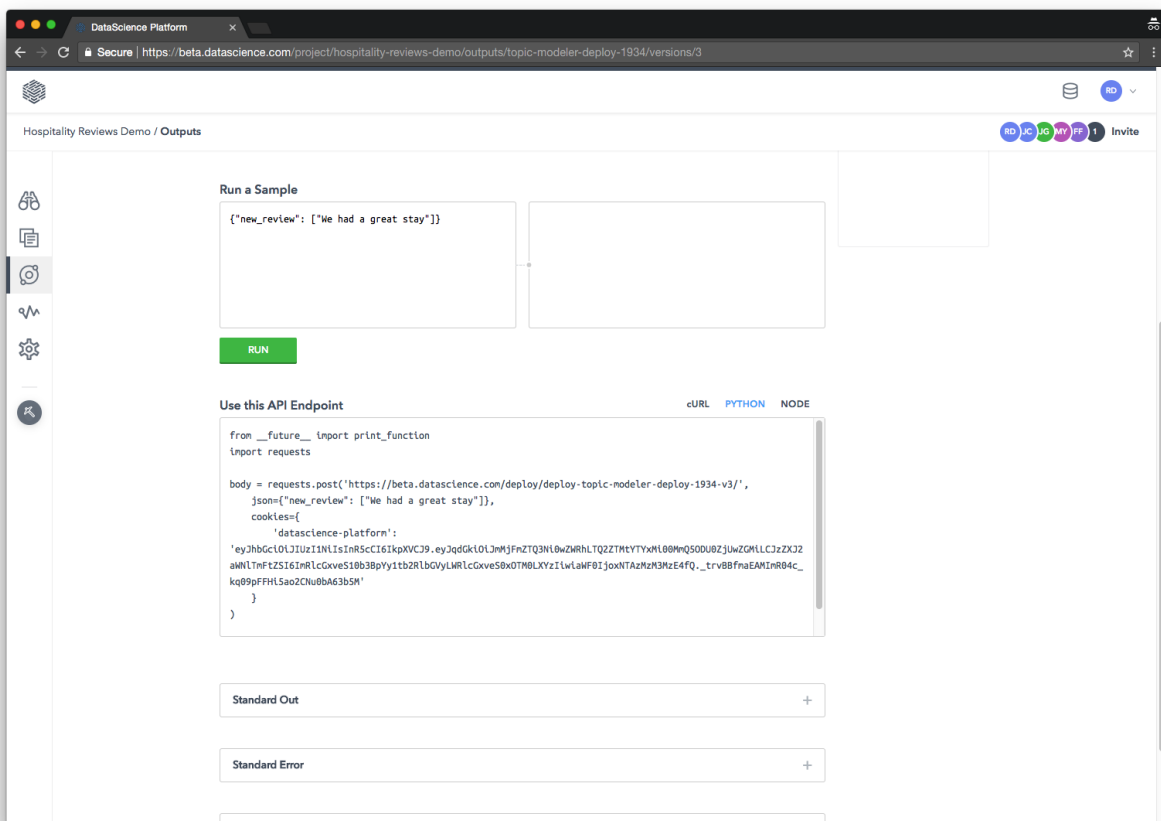
```
[We had a lovely stay ... The food is great but I wish there were more
choices ... The pool was too crowded for me, although the service
was perfect]
```

The call to the API needs a valid deployed model URL, cookie, and model input in JSON format. An example call is already pre-written for you in the Versions tab:

Note that here we are passing a list of reviews to the API as a batch. The deploy function `label\_review` will process this input and return a list of output labels.

The request returns a string of labels as the response, stored in the variable `body`:

```
# call the model API with reviews array
url = 'https://my.datascience.com/deploy/my-deployed-model-v1/'
body = requests.post(url,
```



```

        json={"new_review": reviews},
        cookies={'datascience-platform':
↪ 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪ eyJqdGkiOiI5OGQ4YzU5Mi1kNzFkLTQ4ZWVtYmVlNC0zYWFiNzNiNmFkYTQiLCJzZXJ2aWNlTmFtZSI6ImRlcGxveS10b3BpYy.
↪ LM5YemhQjke342mbBaU171o'
        })

labels = body.text.split(',')

```

### 13.2.2.7 Saving the Output

Finally, we can store the reviews and labels on an AWS S3 location:

```

# output to save to s3
out = zip(reviews, labels)

with open('../labeled_sample_reviews.txt', "w") as out_file:
    out_file.write(str(out))

# save to s3
write_obj_to_s3(out, 'labeled_reviews.txt', 'my_path_to_s3/labeled_reviews.txt')
print('Processed %s reviews'%len(reviews))

```

This process of reading, processing, and scoring new reviews can be run as a nightly *Scheduled Job*. The batch scoring job can write the output to a data store used by your BI tools on a regular schedule. For example, reviews labeled by the API can be powering a dashboard summarizing problematic reviews and displaying review category trends.

## 13.2.3 Deploying a Network Intrusion Prediction API

This example explores how to use the DataScience.com Platform to build a network intrusion detection system with SMS alerts and a reporting front end.

A network intrusion attack is the use of a network that compromises its stability or security. There is a large variety of actions on a network that could be considered as an intrusion. For example:

- A Denial of Service attack, in which the network is overwhelmed with requests, causing other services to become unavailable.
- An attacker searching for hidden files for secrets or sensitive data.

You may look at the signatures of user behavior on a network and try to identify patterns that indicate an attack.

This example will use the [1999 KDD Cup dataset](#).

After some feature engineering, model testing and evaluation, train a gradient boosting classifier, serialize the model along with some metadata, and save it to the project:

```

from sklearn.model_selection import GridSearchCV
cv = GridSearchCV(
    GradientBoostingClassifier(),
    {
        'min_samples_split':[2, 4, 8],
        'max_depth':[2, 3, 4],
        'max_features':[None, 'auto']
    },
    n_jobs=4

```

```

)

cv.fit(X_train, y_train)

model = {
    'model':cv.best_estimator_,
    'features': X_train.columns.values.tolist()
}

import pickle

with open('my-model', 'wb') as model_path:
    pickle.dump(model, model_path)

```

With this model that can predict intrusions, we want to use it for a system that can:

- Read connections in real time and alert team members if an attack is detected.
- If an attack is detected, provide a report indicating why the model believes an attack is underway.

This tutorial will walk through the alert system. This system is a deployed model that, when connections are determined to be an intrusion, sends SMS messages to team members with a link to a report about the attack. The alert builds a custom link by encoding feature values in the URL to the report application. The report application, deployed on Heroku in this example, will use the values of the form to generate an explanation. Then SMS messages are sent via the [Twilio API](#). Here is the alert system:

```

# my app, deploy this script
import os
import yaml
import gzip
import pickle
import os
import pandas as pd
import numpy as np
import sys
import json
from twilio.rest import Client

#read in Twilio credentials from environment variables
account_sid = os.environ['TWILIO_SID']
auth_token = os.environ['TWILIO_TOKEN']
client = Client(account_sid, auth_token)
from_number = os.environ['ALERT_SOURCE_NUMBER']
to_number = os.environ['ALERT_DST_NUMBER']

# collect metadata about runtime to ensure pickle protocol
# is correct, and paths to local dependencies are absolute.
PY_VERSION = sys.version_info.major
APP_PATH = os.path.dirname(os.path.abspath(__file__))

def alert(msg, to_number):
    """Uses the twilio API to send messages to a phone number"""
    message = client.api.account.messages.create(to=to_number,
                                                from_=from_number,
                                                body=msg)

def scale_data(array):
    """Scales the data for the model"""
    if len(array.shape) == 1:
        return StandardScaler().fit_transform(array[:, np.newaxis])

```

```

        else:
            return StandardScaler().fit_transform(array)
def load_config(config_path):
    """Loads a configuration yaml file"""
    with open(config_path) as config_file:
        config = yaml.load(config_file.read())
    return config

def load_model(model_path):
    """Reads the serialized model"""
    with open(model_path, 'rb') as model_file:
        model = pickle.load(model_file)
    return model

def load_data(data_path):
    return pd.read_csv(data_path)

print("Files in APP", os.listdir(APP_PATH))

#Paths to local dependencies
MODEL_DIR = os.path.join(APP_PATH, 'models')
DATA_DIR = os.path.join(APP_PATH, 'data')
MODEL_PATH = os.path.join(MODEL_DIR, 'intruder-model-{}.pkl'.format(PY_VERSION))
DATA_PATH = os.path.join(DATA_DIR, 'intruder-data.csv')
CONFIG_PATH = os.path.join(APP_PATH, 'config.yml')
STATIC_DIR = os.path.join(APP_PATH, 'static')
INDEX_PAGE_PATH = os.path.join(STATIC_DIR, 'index.html')
FEATURES_PATH = os.path.join(MODEL_DIR, 'intruder-model-features-{}.pkl'.format(PY_
↪VERSION))

# load configuration file
config = load_config(CONFIG_PATH)
# model is loaded from local file
model = load_model(MODEL_PATH)
# load list of features
features = load_model(FEATURES_PATH)

classes = model.classes_.tolist()

class_pretty_names = {
    'normal': "Normal Activity",
    'u2r': "User to Root Attack",
    'r2l': "Remote to Local Attack",
    'probe': "Network Probe",
    'dos': "Denial of Service Attack",
}

# Send initial alert, indicating system is online
message = alert("Intruder alert system now online", to_number)

def get_url_from_row(row):
    """This function is used to take a connection (row of data), and build
    a URL to our reporting app. The feature values will fill a form that the report_
    ↪uses to generate an explanation. This app is deployed on Heroku."""

    base = 'https://secure-scrubland-78676.herokuapp.com/individual?'
    args = 'src_bytesname={0}&src_dst_rationame={1}&logged_inname={2}&dst_bytesname=
    ↪{3}&same_srv_ratename={4}&is_flag_S0name={5}&error_ratename={6}&rerror_ratename={7}
    ↪&srv_error_ratename={8}&is_service_FTPname={9}&srv_countname={10}&countname={11}'.
    ↪format(*row)

```

```

    return base + args

def predict(connection):
    """ The function to be deployed. We allow the user to manually set the
        number to which we send SMS messages. The connection (row of data) is sent as the_
        ↪key-values in a dictionary, where keys indicate the feature name,
        and values correspond to feature values."""

    if 'alertnumber' in connection:
        sent_to_number = connection['alertnumber']
    else:
        sent_to_number = to_number

    row = [connection[feature] for feature in features]

    prediction = model.predict(row)[0]

    if prediction == 'normal':
        return {'message': "Normal Activity"}
    else:
        inspect_url = get_url_from_row(list(row))

        alert("{0} detected. To see why, go to {1}".format(class_pretty_
        ↪names[prediction], inspect_url), sent_to_number)
        return {'message': "{} detected, sending alert.".format(class_pretty_
        ↪names[prediction])}

```

With the intruder alert system built and saved as `intruder-predictor.py`, deploy it on the DataScience.com Platform (see our articles on [Deploying APIs](#) for more detailed instructions):

With the model running, you may submit an example to ensure it's working properly:

Because the model detected an attack, an SMS was sent with a report explaining the attack.

## 13.2.4 Deploying an XGBoost Model

In this article, you will learn how to deploy an [XGBoost](#) model on the Platform.


This example will utilize the [Lending Club dataset from Kaggle](#) to illustrate how you can use the Platform's deployed API functionality. The purpose of the model is to identify the loans that are going to default. It's a classification problem for which XGBoost is well-suited.

In a nutshell, XGBoost is a distributed gradient boosting library. XGBoost provides a parallel tree boosting algorithm that is quite fast. For more details about XGBoost, visit [XGBoost documentation](#).

### 13.2.4.1 The Business Use Case

Lending Club is a peer-to-peer online lending platform where individuals can get approved for loans. These loans are broken down in \$25 notes that can be purchased by investors on the Lending Club platform.

Before purchasing notes, investors have access to a variety of information about the loan (such as loan purpose, amount, interest rate, etc.) and the credit history of the borrower (income, delinquencies, home ownership, number of credit lines opened, etc.). The purpose of the model in this example is to predict the probability that a given loan will default before reaching maturity. Investors can then avoid these bad notes and focus on the ones less likely to default.

  
**DEPLOY**

**Intruder Predictor** CHANGE

Branch:  commit 654f660

Path to Model (Required, file must be in root directory)

Environment Configuration:

Language  Dependency Collection

Pip Requirements File:

Compute Resource

Specify Function:

Give Example Data:

Add a Commit Message (Required):

CANCEL DEPLOY

## Version 12

added a verified number

Created: Friday, 7/14/2017 @ 12:11 PM, PDT  
User: aaron kramer  
Status: ■ In Progress: Running  
Function: predict  
Script: intruder\_predictor.py  
Branch: master commit 35675c5  
Language: Python 3  
Dependency Collection: Standard  
Compute Resource: 3.8 GB, 2 CPUs (t2.medium)  
Additional Dependencies: requirements.txt

### Run a Sample

```
{"connection":{"src_bytes": 1, "src_dst_ratio": 1,
"logged_in": 1, "dst_bytes": 1, "same_srv_rate": 1,
"is_flag_S0": 1, "serror_rate": 1, "error_rate":
10, "srv_serror_rate": 1, "is_service_FTP": 1,
"srv_count": 1, "count": 1000,"srv_serror_rate":1}}
```



```
{ "message": "Denial of Service Attack detected,
sending alert." }
```

RUN



### 13.2.4.2 Loading the Data and Training the Model

In a Jupyter notebook (Python 2 session) on the Platform, start by loading and training the model.

First load the data from the public S3 bucket. Use the function `s3_pull_file()` that we defined in this [Connect to Data Sources](#) page. Put in your AWS keys that are stored in your project environment variables. You will also need these libraries:

```
# Platform Kernels: Python 2,3
# Snippet Libraries: xgboost==0.6, boto3==1.4.4, pandas==0.20.3

import xgboost as xgb
import sys
import boto3
import os
import pickle as pkl
import pandas as pd
```

Next, load the data in this cell:

```
s3_creds={'access_key': os.environ['YOUR_AWS_ACCESS_KEY'],
          'secret_key': os.environ['YOUR_AWS_SECRET_KEY']}
s3_pull_file('ds-site-static-assets', 'ds-examples/loan-risk/data/demo_data.p', './
↪demo_data.p', s3_creds)
loan_data = pkl.load(open('demo_data.p', 'rb'))
```

We've already manipulated the data and performed one-hot encoding of the categorical features. You can explore the training feature dataset and response variables by executing the following cell:

```
print(loan_data['X_train'].head())
print(loan_data['y_train'])
```

For simplicity, make the assumption that you have found the best hyperparameters of your model. Then, train the XGBoost model with the best set of hyperparameters:

```
train_data = xgb.DMatrix(loan_data['X_train'], label=loan_data['y_train'])
param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }
num_round = 2
bst = xgb.train(param, train_data, num_round)
```

Now, test the predict function with the first ten entries of the training set:

```
tmp_test_data = xgb.DMatrix(loan_data['X_train'].head(10))
preds = bst.predict(tmp_test_data)
preds
```

The results you should see are as follows:

```
array([ 0.25122303, 0.4519583 , 0.25122303, 0.15333922, 0.1026786 ,
        0.15333922, 0.1026786 , 0.25122303, 0.25122303, 0.25122303],
      dtype=float32)
```

You've now trained an XGBoost model and are able to use its `predict()` function on a loan dataset. That mimics what would happen in production: you get information about the loan and you want to predict the default probability with a pre-trained model.

Next, serialize the model and write it to disk using the `pickle` library. The model is converted into a byte stream that can be read by the `load()` method of the `pickle` library.

```
# let's write to disk a serialized version of our xgboost model:
pkl.dump(bst, open('xgb_model.pkl', 'wb'))
```

The model is saved to disk.

### 13.2.4.3 Deploying the Model

In a script file we called `xgboost_model_api.py`, we wrote the function `api_predict(data)` we want to deploy (see below). Remember that the data that is being passed as argument to the function is in JSON format. That is why we perform a conversion from JSON to dataframe.

```
# Content of the file 'xgboost_model_api.py'

import xgboost as xgb
import pandas as pd
import os
import sys
import pickle as pkl

# Let's load the pickled xgboost model:
xgb_model_api = pkl.load(open('xgb_model.pkl', 'rb'))

def api_predict(data):
    """This function takes the loan data in a JSON format
    and returns the probability of default based on an XGBoost
    model.

    Parameters
    -----

    loan_data: JSON data structure containing the loan data.

    Returns
    -----

    A dictionary with the default probabilities. The keys correspond to the
    loan IDs.
    """

    # Conversion to a dataframe :
    json_2_df = pd.DataFrame.from_dict(data, orient='index')

    # Re-order the columns to match the column order of the training dataset.
    # This is important. Internally xgboost transform your dataset into the
    # libsvm data format. This format takes the index of the feature and associates
    # a value to the index value. It's a sparse data format. Consequently, preserving
    # feature order is important.
    json_2_df = json_2_df[[u'loan_amnt', u'int_rate', u'dti', u'annual_inc', u'delinq_
→2yrs',
        u'open_acc', u'revol_util', u'term_ 36 months', u'term_ 60 months',
        u'purpose_car', u'purpose_credit_card', u'purpose_debt_consolidation',
        u'purpose_educational', u'purpose_home_improvement', u'purpose_house',
        u'purpose_major_purchase', u'purpose_medical', u'purpose_moving',
        u'purpose_other', u'purpose_renewable_energy',
        u'purpose_small_business', u'purpose_vacation', u'purpose_wedding',
        u'addr_state_AK', u'addr_state_AL', u'addr_state_AR', u'addr_state_AZ',
        u'addr_state_CA', u'addr_state_CO', u'addr_state_CT', u'addr_state_DC',
```

```

u'addr_state_DE', u'addr_state_FL', u'addr_state_GA', u'addr_state_HI',
u'addr_state_IA', u'addr_state_ID', u'addr_state_IL', u'addr_state_IN',
u'addr_state_KS', u'addr_state_KY', u'addr_state_LA', u'addr_state_MA',
u'addr_state_MD', u'addr_state_ME', u'addr_state_MI', u'addr_state_MN',
u'addr_state_MO', u'addr_state_MS', u'addr_state_MT', u'addr_state_NC',
u'addr_state_ND', u'addr_state_NE', u'addr_state_NH', u'addr_state_NJ',
u'addr_state_NM', u'addr_state_NV', u'addr_state_NY', u'addr_state_OH',
u'addr_state_OK', u'addr_state_OR', u'addr_state_PA', u'addr_state_RI',
u'addr_state_SC', u'addr_state_SD', u'addr_state_TN', u'addr_state_TX',
u'addr_state_UT', u'addr_state_VA', u'addr_state_VT', u'addr_state_WA',
u'addr_state_WI', u'addr_state_WV', u'addr_state_WY',
u'home_ownership_ANY', u'home_ownership_MORTGAGE',
u'home_ownership_NONE', u'home_ownership_OTHER', u'home_ownership_OWN',
u'home_ownership_RENT']]

loan_data_dmatrix = xgb.DMatrix(json_2_df)
# Here we are using the predict() method of the XGBoost model to predict
# default on the data passed to the api_predict() function.
res = xgb_model_api.predict(loan_data_dmatrix)

return { "{}".format(loan_id):result for result,loan_id in zip(res.tolist(), json_
↪2_df.index.values.tolist()) }

```

The function `api_predict()` takes data in JSON format. In the case above, we expect the data to have this particular format:

```

{"data":
{"107265":{"loan_amnt":14000.0,"int_rate":17.57,"dti":21.6,"annual_inc":82000.0,
↪"delinq_2yrs":1.0,"open_acc":24.0,"revol_util":43.8,"term_
36 months":1,"term_ 60
months":0,"purpose_car":0,"purpose_credit_card":0,"purpose_debt_consolidation":1,
↪"purpose_educational":0,"purpose_home_improvement":0,"purpose_house":0,
"purpose_major_purchase":0,"purpose_medical":0,"purpose_moving":0,"purpose_other":0,
↪"purpose_renewable_energy":0,"purpose_small_business":0,
"purpose_vacation":0,"purpose_wedding":0,"addr_state_AK":0,"addr_state_AL":0,"addr_
↪state_AR":0,"addr_state_AZ":1,"addr_state_CA":0,
"addr_state_CO":0,"addr_state_CT":0,"addr_state_DC":0,"addr_state_DE":0,"addr_state_FL
↪":0,"addr_state_GA":0,"addr_state_HI":0,
"addr_state_IA":0,"addr_state_ID":0,"addr_state_IL":0,"addr_state_IN":0,"addr_state_KS
↪":0,"addr_state_KY":0,"addr_state_LA":0,
"addr_state_MA":0,"addr_state_MD":0,"addr_state_ME":0,"addr_state_MI":0,"addr_state_MN
↪":0,"addr_state_MO":0,"addr_state_MS":0,
"addr_state_MT":0,"addr_state_NC":0,"addr_state_ND":0,"addr_state_NE":0,"addr_state_NH
↪":0,"addr_state_NJ":0,"addr_state_NM":0,
"addr_state_NV":0,"addr_state_NY":0,"addr_state_OH":0,"addr_state_OK":0,"addr_state_OR
↪":0,"addr_state_PA":0,"addr_state_RI":0,
"addr_state_SC":0,"addr_state_SD":0,"addr_state_TN":0,"addr_state_TX":0,"addr_state_UT
↪":0,"addr_state_VA":0,"addr_state_VT":0,
"addr_state_WA":0,"addr_state_WI":0,"addr_state_WV":0,"addr_state_WY":0,"home_
↪ownership_ANY":0,"home_ownership_MORTGAGE":1,
"home_ownership_NONE":0,"home_ownership_OTHER":0,"home_ownership_OWN":0,"home_
↪ownership_RENT":0}}}

```

The value of the key “data” is the data structure our function is expecting. The index of the data frame will be the keys of the “data” dictionary (`orient=index`).

There is one last step needed before deploying the REST API. You have to create a pip dependency file to capture any libraries needed to deploy your model. Name this file `api_requirements.txt` and put it in the top level

folder of your project. In this case, XGBoost and pandas are needed outside the standard dependency collection. Our file thus contains the following dependencies:

```
xgboost>=0.6
pandas>=0.20.3
```

These packages need to be installed on the REST API Docker container for your model to work. After that step is done, make sure you Sync your work with remote GitHub. This will generate a new commit ID containing the latest changes.

You're now ready to deploy the model as a REST API! The first step in this process is to go to Deploy an API from the Quick Actions button.

Below is a screenshot of the filled out Deploy form. Specify the Python script file you used (`xgboost_model_api.py`) as well as the name of the function you want to deploy (`api_predict()`). Also provide an example of a dataset you want to pass to the API. In this case, you may use the example mentioned above.

The screenshot shows the 'DEPLOY' form for a project named 'lendingclub\_xgboost\_test'. The form is divided into several sections:

- General:** Shows the project name 'lendingclub\_xgboost\_test' and a 'CHANGE' button.
- Version 8:** A sidebar on the left showing a list of versions from 1 to 8.
- Branch:** A dropdown menu set to 'jr\_dev' and a commit ID 'a15e4a2'.
- Path to Model:** A text field containing 'xgboost\_model\_api.py'.
- Environment Configuration:** A section with tabs for 'Language' (Python 2), 'Dependency Collection' (Standard Collection), and 'Pip Requirements File' (api\_requirements.txt). It also shows 'Compute Resource' as '2 GB / 1 CPU'.
- Specify Function:** A text field containing 'api\_predict'.
- Give Example Data:** A text field containing '{"data": {"107265": {"loan\_amnt": 141'.
- Add a Commit Message (Required):** A text field with the placeholder 'Add a commit message'.
- Run a Sample:** A section at the bottom showing a sample JSON output from the API.
- Buttons:** 'CANCEL' and 'DEPLOY' buttons at the bottom right.

The last step is to specify the dependencies of your deployed function. This can be done by clicking the Edit option of the Environment Configuration section. Include the name of your `pip` requirements file. We recommend putting that file in the same folder as the Python script containing your deployed function.

Click Save and your function has now been deployed as a REST API.

Next, take a look at the Versions tab of your API. Below is an example snapshot.

The screenshot shows the deployment configuration page for a function named `Lendingclub_xgboost_test`. The interface includes a sidebar with navigation options like 'General', 'Version 3', 'Logs', 'Status', 'Functions', 'Script', 'Branch', 'Language', 'Dependency Collection', 'Compute Resource', and 'Additional Dependencies'. The main content area is for the 'General' tab, showing the following configuration:

- Branch:** `jr_dev` (commit `a15e4a2`)
- File:** `xgboost_model_api.py`
- Language:** `Python 2` (dropdown menu)
- Dependency Collection:** `Standard` (dropdown menu)
- Add Requirements (optional):**
  - PIP:** `api_requirements.txt`
  - R:** Enter path to requirements file
  - APT:** Enter path to requirements file
- Compute Resource:** `2 GB / 1 CPU` (dropdown menu)

At the bottom, there is a 'Run a Sample' section with a code editor containing a list of address state coordinates. A blue 'SAVE' button is located at the bottom right of the configuration area. On the right side, there is a 'DEPLOY NEW VERSION' button and a list of previous versions (Version 1 through Version 6).

#### 13.2.4.4 Conclusion

You now have a deployed function that predicts the default probability of a loan available on the Lending Club platform. You can use this API endpoint in a variety of applications. We show three different ways to call this API using `cURL`, `Python`, and `Node`.

A colleague can call your model within their `Python` code or a front-end web developer can call the API using `Node` and create a web-based app that allows institutional investors to select loans to purchase.

*How to Create and Deploy a Shiny App:* Shiny by RStudio is a framework for turning R code into interactive dashboards. You can build RShiny apps on the DataScience.com Platform within the familiar R or RStudio environment. This quick tutorial shows how to build an interactive map with RShiny using a dataset on Uber pickups in New York City.

*Using a Deployed API:* The purpose of this article is to show an example of a use case for a deployed API. An API can be used to automatically score incoming data with a pre-trained deployed model. In this article, the deployed API will take hotel reviews as input and assign category labels. Labeled reviews can then be sorted, summarized with a BI tool, and addressed by appropriate departments.

*Deploying a Network Intrusion Prediction API:* This example explores how to use the DataScience.com Platform to build a network intrusion detection system with SMS alerts and a reporting front end.

*Deploying an XGBoost Model:* In this article, you will learn how to deploy an XGBoost model on the Platform to predict loan repayment default in peer-to-peer lending platforms. This example will utilize the Lending Club dataset from Kaggle to illustrate how you can use the Platform's deployed API functionality.

# Lendingclub\_xgboost\_test

name says it all

[General](#)
[Versions](#)
[More Options](#)

## Version 8

tmp

Created: Thursday, 8/17/2017 @ 6:54 PM, PDT  
 User: JR Gauthier  
 Status: ● In Progress: Running  
 Function: api\_predict  
 Script: xgboost\_model\_api.py  
 Branch: jr\_dev commit a15e4a2  
 Language: Python 2  
 Dependency Collection: Standard  
 Compute Resource: 2 GB, 1 CPU  
 Additional Dependencies: api\_requirements.txt

## Run a Sample

```
{ "data": { "107265":
{"loan_amnt":14000.0,"int_rate":17.57,"dti":21.6
,"annual_inc":82000.0,"delinq_2yrs":1.0,"open_ac
c":24.0,"revol_util":43.8,"term_ 36
months":1,"term_ 60
months":0,"purpose_car":0,"purpose_credit_card":
0,"purpose_debt_consolidation":1,"purpose_educat
ional":0,"purpose_home_improvement":0,"purpose_h
```

```
{ "107265": 0.25122302770614624 }
```

RUN

## Use this API Endpoint

[cURL](#)
[PYTHON](#)
[NODE](#)

```
curl -X POST -d '{ "data": { "107265":
{"loan_amnt":14000.0,"int_rate":17.57,"dti":21.6,"annual_inc":82000.0,"delinq_2yrs":1.0,"open_acc":24.0,"
revol_util":43.8,"term_ 36 months":1,"term_ 60
months":0,"purpose_car":0,"purpose_credit_card":0,"purpose_debt_consolidation":1,"purpose_educational":0,
"purpose_home_improvement":0,"purpose_house":0,"purpose_major_purchase":0,"purpose_medical":0,"purpose_mo
ving":0,"purpose_other":0,"purpose_renewable_energy":0,"purpose_small_business":0,"purpose_vacation":0,"p
urpose_wedding":0,"addr_state_AK":0,"addr_state_AL":0,"addr_state_AR":0,"addr_state_AZ":1,"addr_state_CA"
:0,"addr_state_CO":0,"addr_state_CT":0,"addr_state_DC":0,"addr_state_DE":0,"addr_state_FL":0,"addr_state_
GA":0,"addr_state_HI":0,"addr_state_IA":0,"addr_state_ID":0,"addr_state_IL":0,"addr_state_IN":0,"addr_sta
te_KS":0,"addr_state_KY":0,"addr_state_LA":0,"addr_state_MA":0,"addr_state_MD":0,"addr_state_ME":0,"addr_
state_MI":0,"addr_state_MN":0,"addr_state_MO":0,"addr_state_MS":0,"addr_state_MT":0,"addr_state_NC":0,"ad
dr_state_ND":0,"addr_state_NE":0,"addr_state_NH":0,"addr_state_NJ":0,"addr_state_NM":0,"addr_state_NV":0,
"addr_state_NY":0,"addr_state_OH":0,"addr_state_OK":0,"addr_state_OR":0,"addr_state_PA":0,"addr_state_RI"
:0,"addr_state_SC":0,"addr_state_SD":0,"addr_state_TN":0,"addr_state_TX":0,"addr_state_UT":0,"addr_state
```

## CHAPTER 14

---

### How to Read These Docs

---

In the User docs, you'll learn how to set up an account and launch different types of analyses across the Platform.





## CHAPTER 15

---

### Just Getting Started with the Platform?

---

If you're just starting out on the Platform, consider going through the onboarding videos and exercises contained in our self-guided learning module. This curated tour of the Platform covers all the key features for data scientist users. To request access, contact [success@datascience.com](mailto:success@datascience.com).