**Document Title:** Cloudera Data Science lab guide  
**Date:** October 2018

**Version:** 2.4

**Authors**: Colm Moynihan, Toby Ferguson, Andre Molenaar

**Cloudera Data Science Workbench Labs**

**Table of Contents**

[Lab 11 – Experiments 2](#_Toc9873797)

[Lab 12 – Working with Models 9](#_Toc9873798)

## Lab 11 – Experiments

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to run batch experiments that track different versions of code, input parameters, and output (both metrics and files).

**Challenge**

As data scientists iteratively develop models, they often experiment with datasets, features, libraries, algorithms, and parameters. Even small changes can significantly impact the resulting model. This means data scientists need the ability to iterate and repeat similar experiments in parallel and on demand, as they rely on differences in output and scores to tune parameters until they obtain the best fit for the problem at hand. Such a training workflow requires versioning of the file system, input parameters, and output of each training run.

Without versioned experiments you would need intense process rigor to consistently track training artifacts (data, parameters, code, etc.), and even then it might be impossible to reproduce and explain a given result. This can lead to wasted time/effort during collaboration, not to mention the compliance risks introduced.

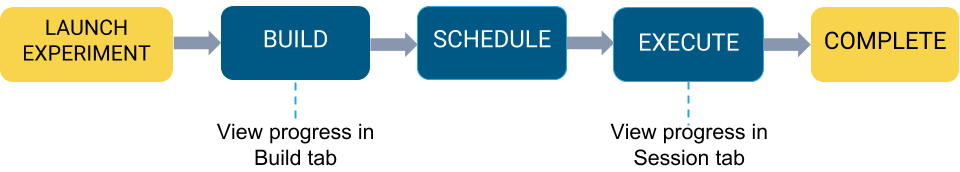
**Solution**

Starting with version 1.4, Cloudera Data Science Workbench uses experiments to facilitate ad-hoc batch execution and model training. Experiments are batch executed workloads where the code, input parameters, and output artifacts are versioned. This feature also provides a lightweight ability to track output data, including files, metrics, and metadata for comparison.

**Concepts**

The term experiment refers to a non interactive batch execution script that is versioned across input parameters, project files, and output. Batch experiments are associated with a specific project (much like sessions or jobs) and have no notion of scheduling; they run at creation time. To support versioning of the project files and retain run-level artifacts and metadata, each experiment is executed in an isolated container.

**Lifecycle of an Experiment**



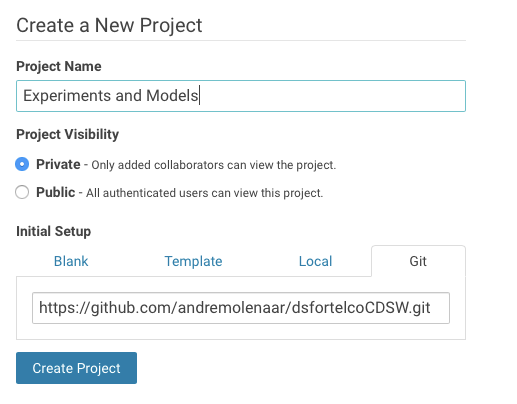
**Step 1: Create a new project**

Go to the homepage of your Data Science workbench, and create a ‘New’ project.

Call the new repository something like Experiments and Models.

Create the repository as a clone of the github repository:

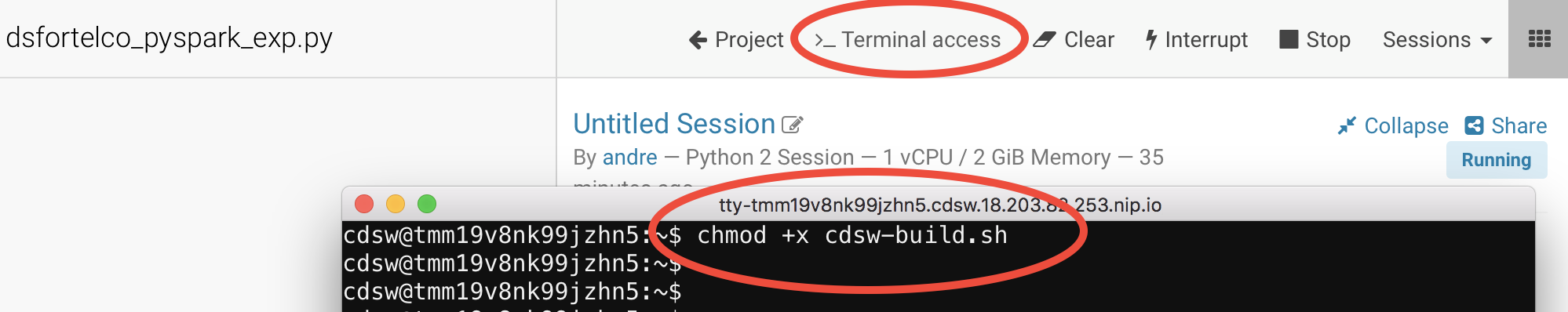
<https://github.com/andremolenaar/dsfortelcoCDSW.git>



Start a workbench with a Python 2 and 2 GB of memory.

When the workbench is available, open a terminal window and make the cdsw-build.sh program executable. Use the following command to do that:

chmod +x cdsw-build.sh



**Step 2: Examin dsfortelco\_sklearn\_exp.py**

Open the file “dsfortelco\_sklearn\_exp.py”. This is a python program that builds a churn model to predict customer churn (the likelyhood that this customer is going to stop his subscription with his telecom operator). There is a dataset available on hdfs (/tmp/churn\_all.csv), with customer data, including a churn indicator field.

The program is going to build a churn prediction model using the Random Forest algorithm. Random forests are ensembles of decision trees. Random forests are one of the most successful machine learning models for classification and regression. They combine many decision trees in order to reduce the risk of overfitting. Like decision trees, random forests handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.  
  
spark.mllib supports random forests for binary and multiclass classification and for regression, using both continuous and categorical features. spark.mllib implements random forests using the existing decision tree implementation. Please see the decision tree guide for more information on trees.

The Random Forest algorithm expects a couple of parameters:

* numTrees: Number of trees in the forest.  
  Increasing the number of trees will decrease the variance in predictions, improving the model’s test-time accuracy.  
  Training time increases roughly linearly in the number of trees.
* maxDepth: Maximum depth of each tree in the forest.  
  Increasing the depth makes the model more expressive and powerful. However, deep trees take longer to train and are also more prone to overfitting.  
  In general, it is acceptable to train deeper trees when using random forests than when using a single decision tree. One tree is more likely to overfit than a random forest (because of the variance reduction from averaging multiple trees in the forest).

In the dsfortelco\_pyspark\_exp.py program, these parameters can be passed to the program at runtime. In the lines 38 and 39, these parameters are passed to python variables:

param\_numTrees=int(sys.argv[1])

param\_maxDepth=int(sys.argv[2])

Also note that at the lines 69 and 70, the quality indicator for the Random Forest model, are written back to the Data Science Workbench repository:

cdsw.track\_metric("auroc", auroc)  
cdsw.track\_metric("ap", ap)

These indicators will show up later in the Experiments dashboard.

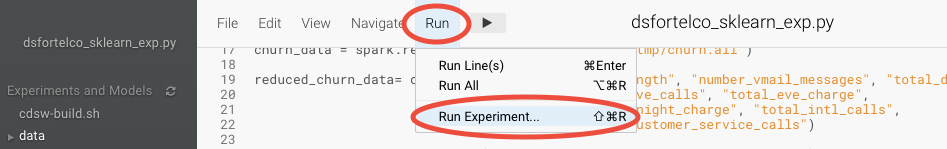
**Step 3: Run the experiment for the first time**

Now, run the experiment using the following parameters:

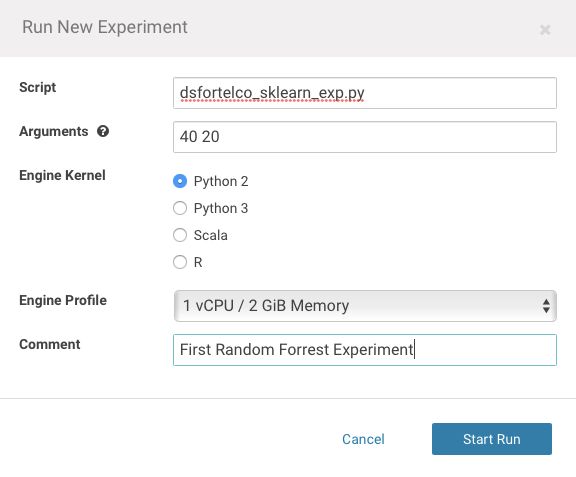
numTrees = 40

numDepth = 20

From the menu, select Run -> Experiments.



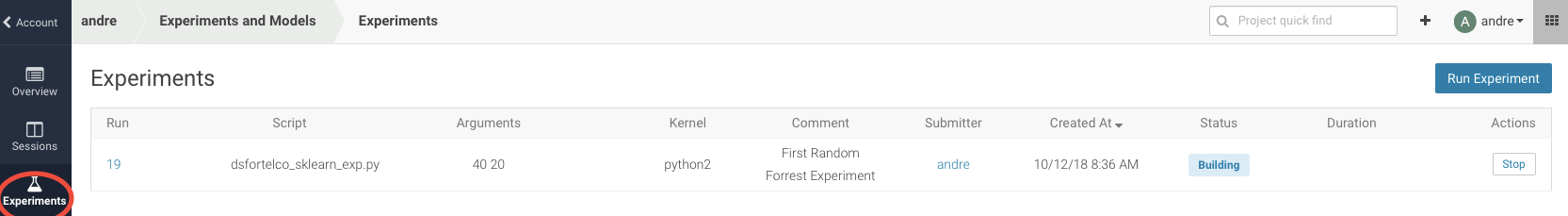
Specify the arguments for this run, by typing the numbers behind the arguments field. Note that these fields are separated by a space and that there is no comma (,)



Now, in the background, the Data Science Workbench environment will spin up a new docker container, where this program will run.

**Step 4: Check the results for the first experiment**

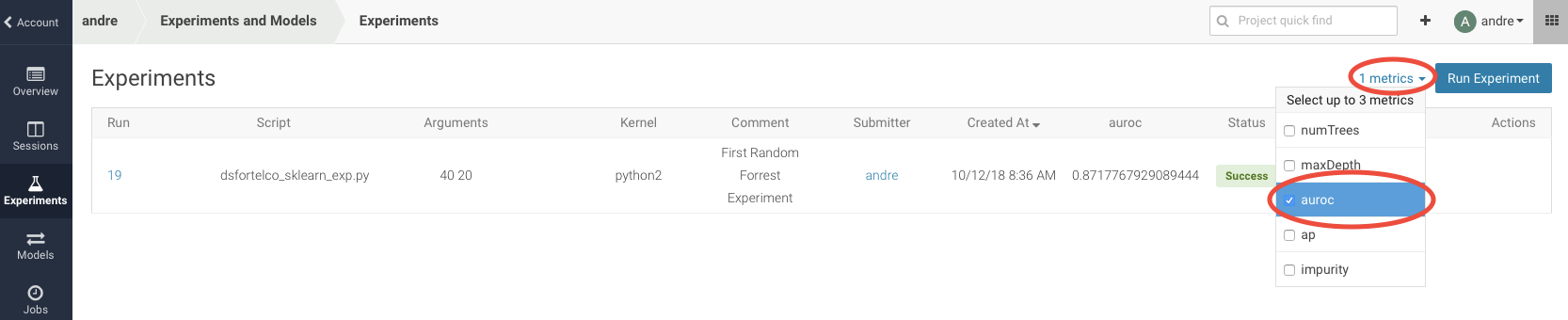
Go back to the ‘Projects’ page in CDSW, and hit the ‘Experiments’ button.



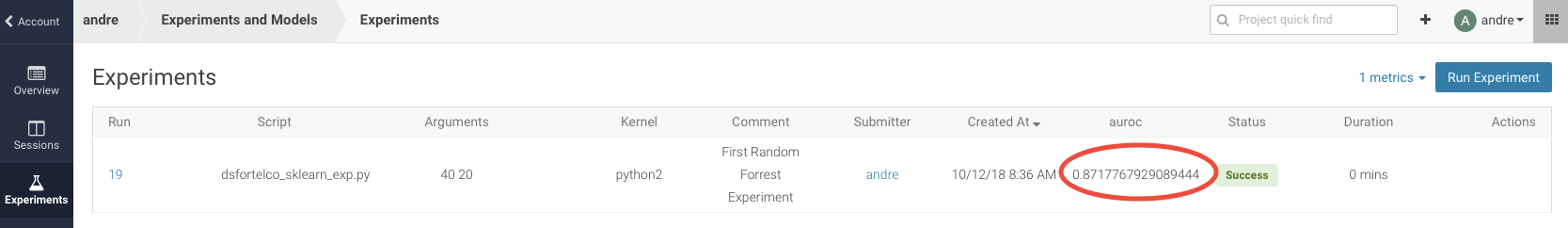
If the Status indicates ‘Running’, you have to wait till the run is completed.

In case the status is ‘Build Failed’ or ‘Failed’, check the log information. This is accessible by clicking on the run number of your experiments. There you can find the session log, as well as the build information.

In case your status indicates ‘Success’, you should be able to see the auroc (Area Under the Curve) model quality indicator. It might be that this value is hidden by the CDSW user interface. in that case, click on the ‘3 metrics’ links, and select the auroc field. It might be needed to de-select some other fields, since the interface can only show 3 metrics at the same time.



When the auroc metric is selected, you will be able to see the value.



In this example, 0.871

Not bad, but maybe there are better hyper parameter values available.

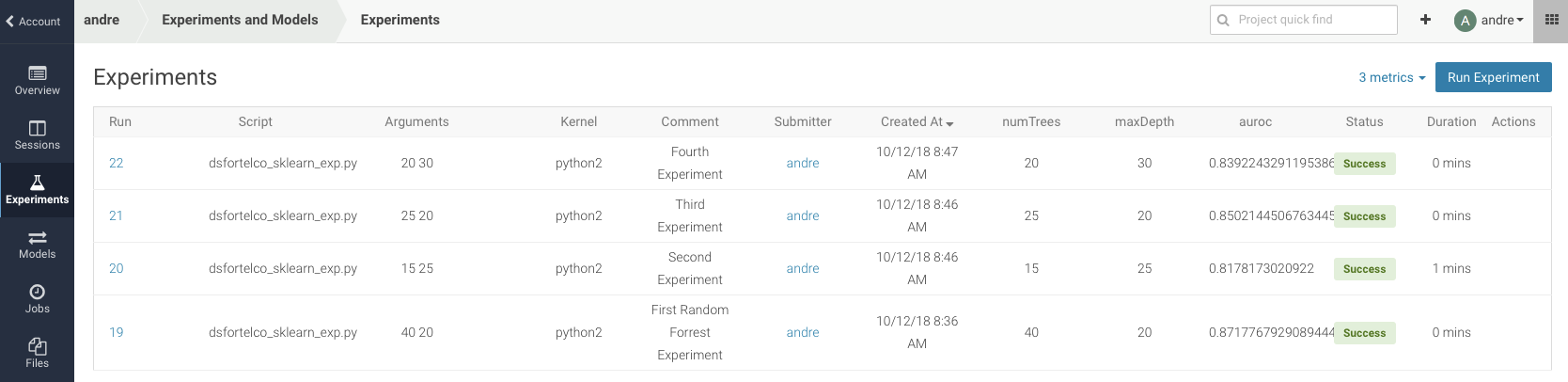
**Step 5: Re run the experiment several times**

Now, re-run the experiment 3 more times and try different values for NumTrees and NumDepth.

Try the following values:

|  |  |
| --- | --- |
| NumTrees | NumDepth |
| 15 | 25 |
| 25 | 20 |
| Try something yourself | Try something yourself |

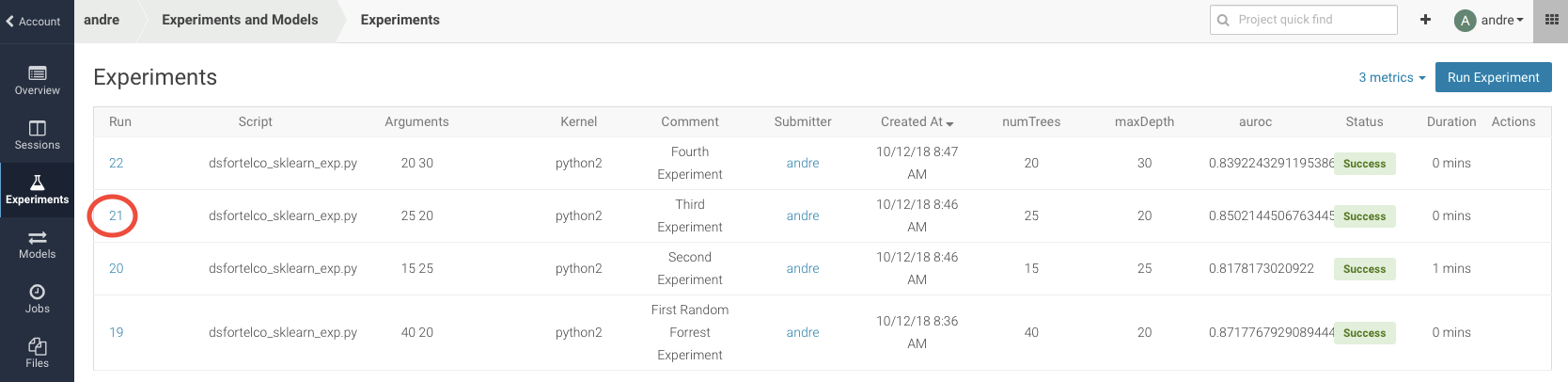
When all runs have completed successfully, check which parameters had the best quality (best predictive value). This is represented by the highest ‘area under the curve’, auroc metric.



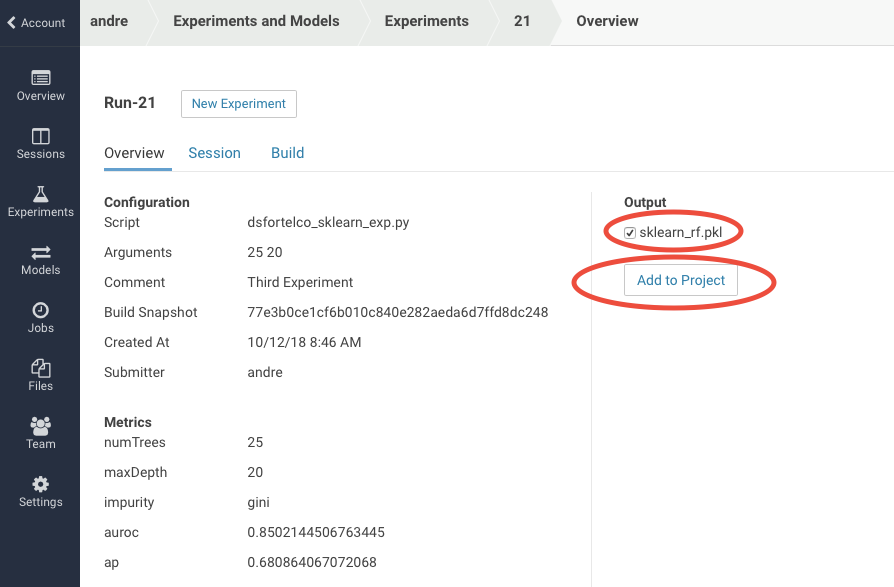
In this example, run 21 had the highest auroc value, so that is the model that you would want to use for your business.

**Step 6: Save the best model to your environment**

Select the run number with the best predictive value, in this example, run number 21.



In the Overview screen of the experiment, you can see that the model in spark format, is captured in the file ‘sklearn\_rf.pkl’. Select this file and hit the ‘Add to Project’ button. This will copy the model to your project directory.



## Lab 12 – Working with Models

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to build, deploy, and manage models as REST APIs to serve predictions.

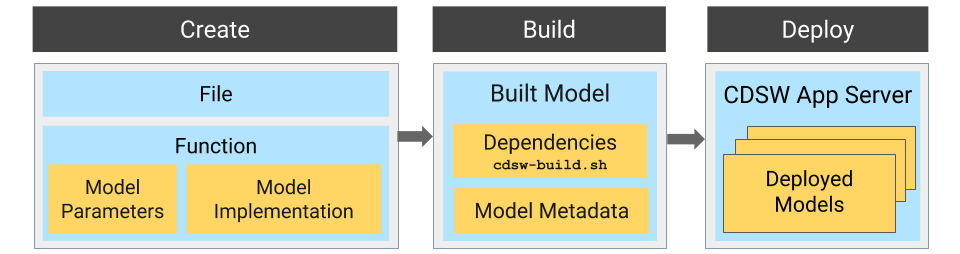
**Challenge**

Data scientists often develop models using a variety of Python/R open source packages. The challenge lies in actually exposing those models to stakeholders who can test the model. In most organizations, the model deployment process will require assistance from a separate DevOps team who likely have their own policies about deploying new code.  
  
For example, a model that has been developed in Python by data scientists might be rebuilt in another language by the devops team before it is actually deployed. This process can be slow and error-prone. It can take months to deploy new models, if at all. This also introduces compliance risks when you take into account the fact that the new re-developed model might not be even be an accurate reproduction of the original model.  
  
Once a model has been deployed, you then need to ensure that the devops team has a way to rollback the model to a previous version if needed. This means the data science team also needs a reliable way to retain history of the models they build and ensure that they can rebuild a specific version if needed. At any time, data scientists (or any other stakeholders) must have a way to accurately identify which version of a model is/was deployed.

**Solution**

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to build and deploy their own models as REST APIs. Data scientists can now select a Python or R function within a project file, and Cloudera Data Science Workbench will:

* Create a snapshot of model code, model parameters, and dependencies.
* Package a trained model into an immutable artifact and provide basic serving code.
* Add a REST endpoint that automatically accepts input parameters matching the function, and that returns a data structure that matches the function’s return type.
* Save the model along with some metadata.
* Deploy a specified number of model API replicas, automatically load balanced.

Stages of the Model Deployment Process  
  
  
  
  
  
**Step 1: Examine the program predic\_churn\_sklearn.py**

Open the project you created in the previous lab, and examine the file.

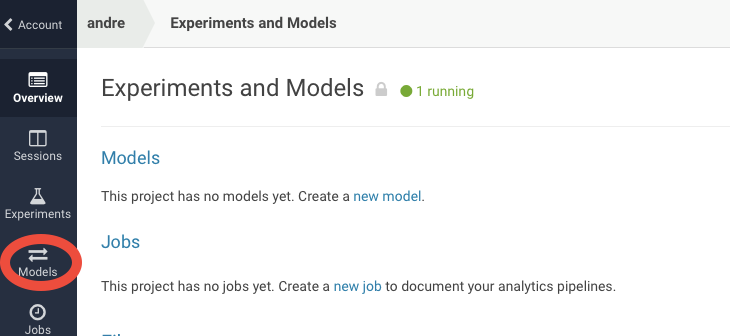


This PySpark program uses the pickle.load mechanism to deply models.. The model it refers to the sklearn\_rf.pkl file, was saved in the previous lab from the experiment with the best predictive model.

There is a predict definition which is the function that calls the model, using features, and will return a result variable.

**Step 2: Deploy the model**

From the projects page of your project, select the ‘Models’ button.



Select ‘New Model’, and populate specify the following configuration:

Name: something like “My Churn Prediction Model”

Description: Anything you want

File: predict\_churn\_sklearn.py

Function: predict

Example Inp: {

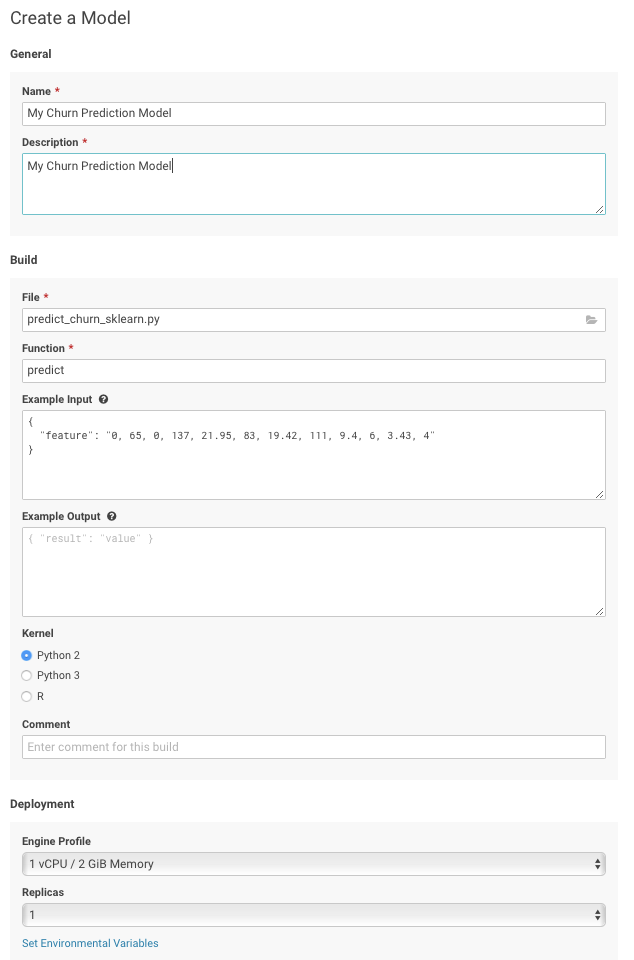
"feature": "0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4"

}

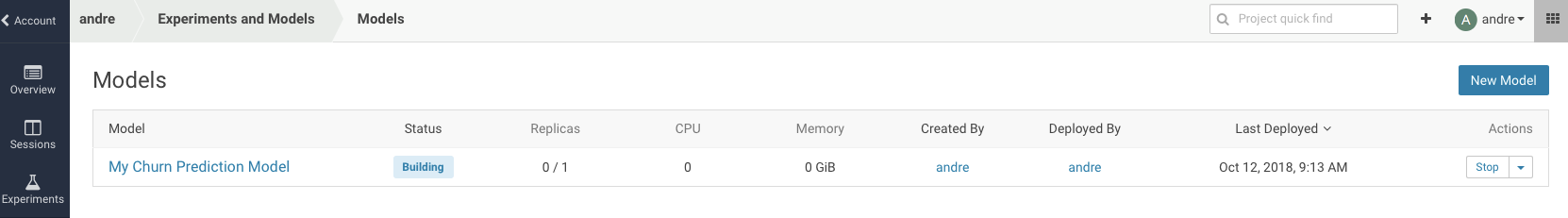
Kernal: Python 2

Engine: 1 vCPU / 2 GiB Memory

Replicas: 1

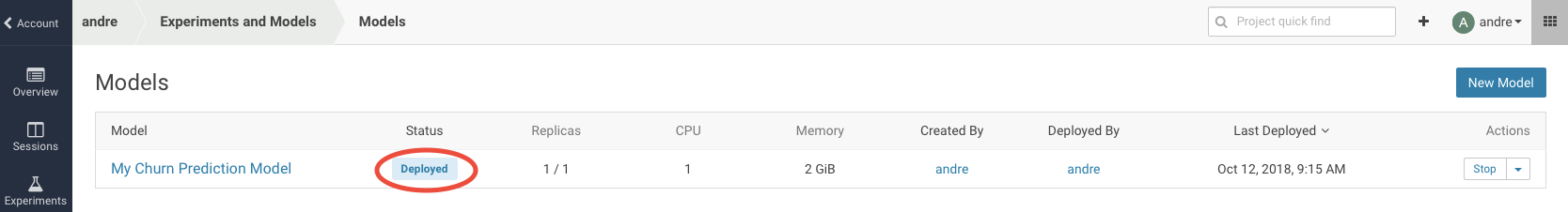


If all parameters are set, you can hit the ‘Deploy Model’ button. Wait till the model is deployed. This will take several minutes.

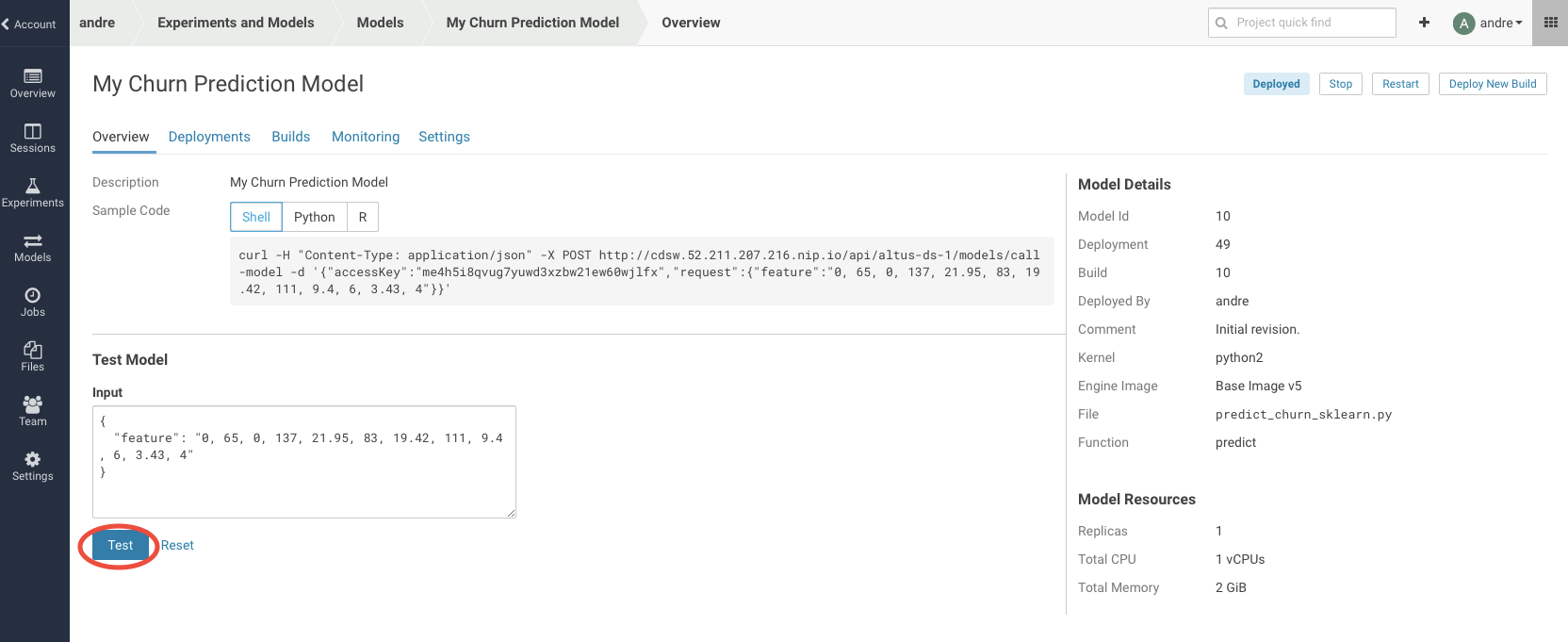


**Step 3: Test the deployed model**

After the several minutes, your model should get to the ‘Deployed’ state.



Now, click on the Model Name link, to go to the Model Overview page. From the that page, hit the ‘Test’ button to check if the model is working.



If your model is working, you should receive an output similar like this:



The green color with success is telling that our REST call to the model is technically working. And if you examine the response: {“result”: 1}, it returns a 1, which mean that customer with these features is likely to churn.

Now, lets change the input parameters and call the predict function again. Put the following values in the Input field:

{

"feature": "0, 95, 0, 88, 26.62, 75, 21.05, 115, 8.65, 5, 3.32, 3"

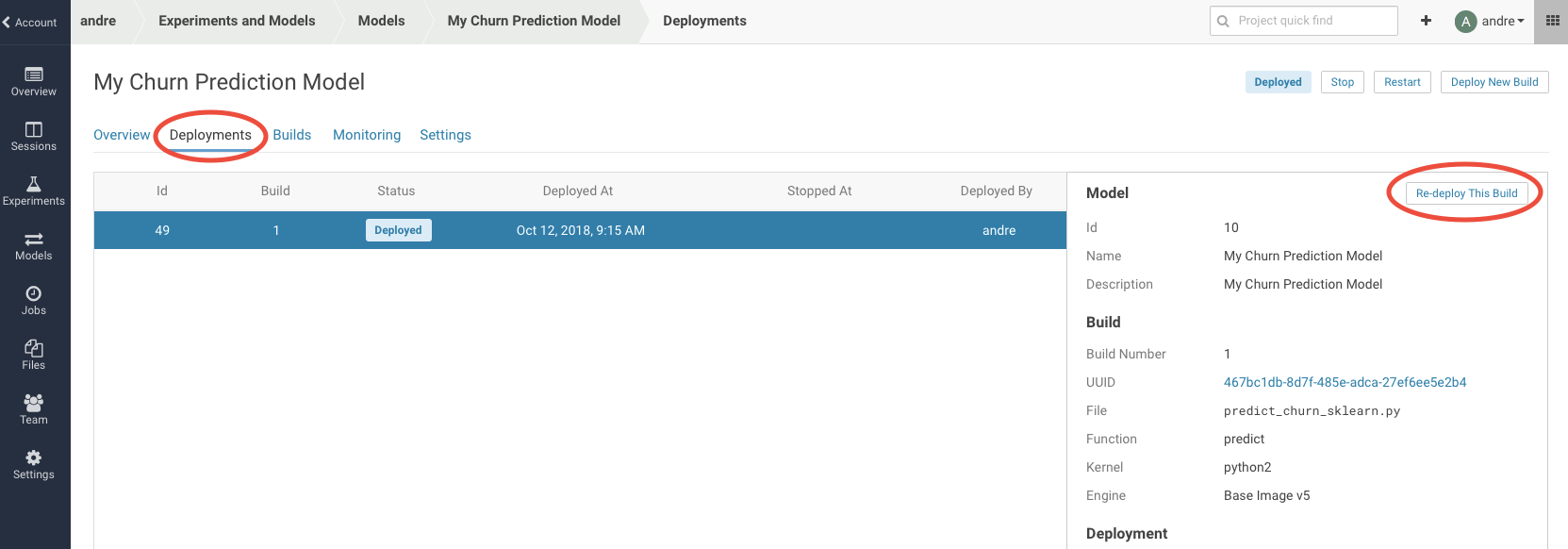
}



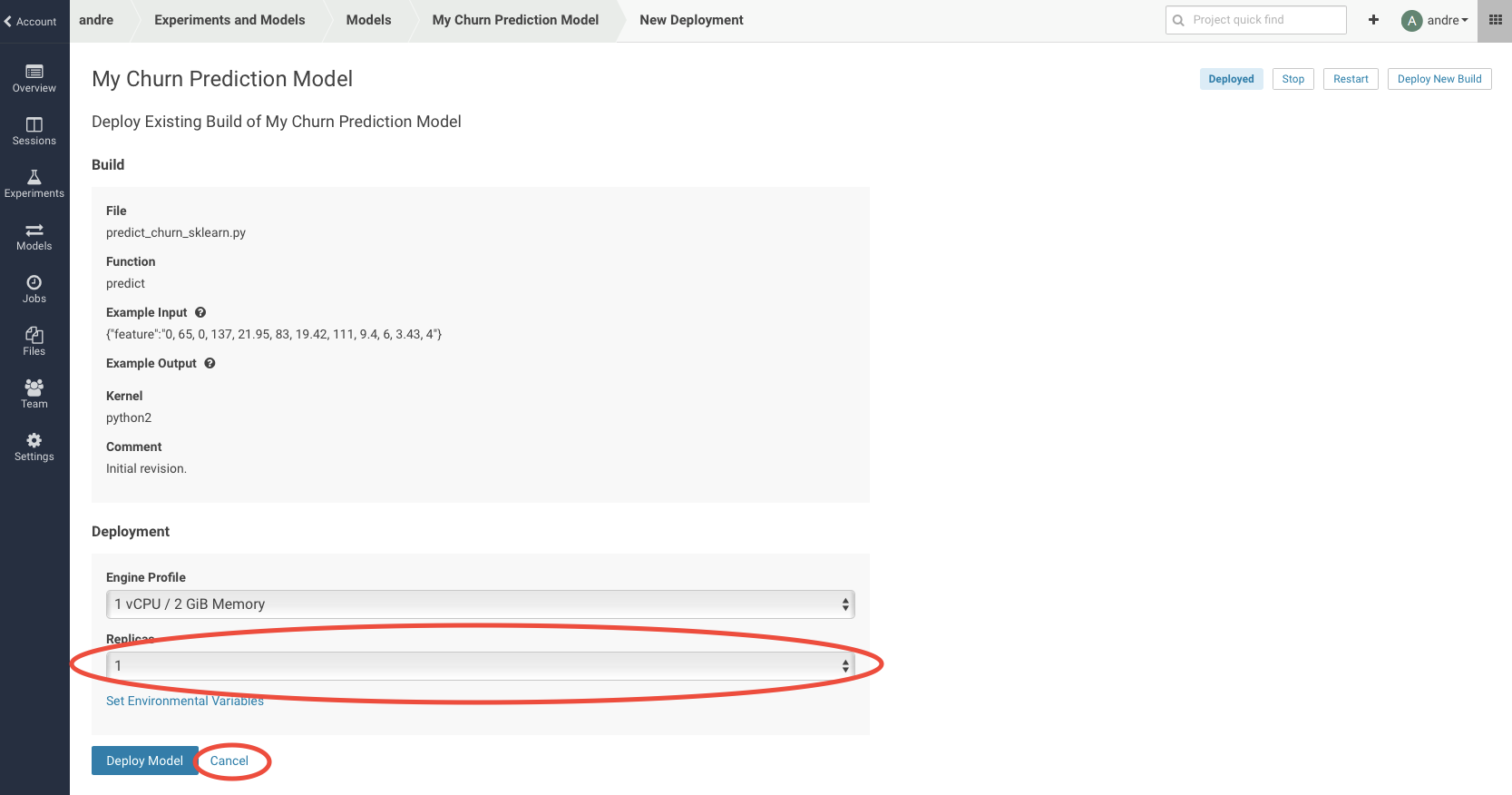
With these input parameters, the model returns 0, which mean that the customer is not likely to churn.

**Step 4: Model Administration**

When a model is deployed, Cloudera Data Science Workbench allows you to specify a number of replicas that will be deployed to serve requests. For each active model, you can monitor its replicas by going to the model's Monitoring page. On this page you can track the number of requests being served by each replica, success and failure rates, and their associated stderr and stdout logs. Depending on future resource requirements, you can increase or decrease the number of replicas by re-deploying the model.

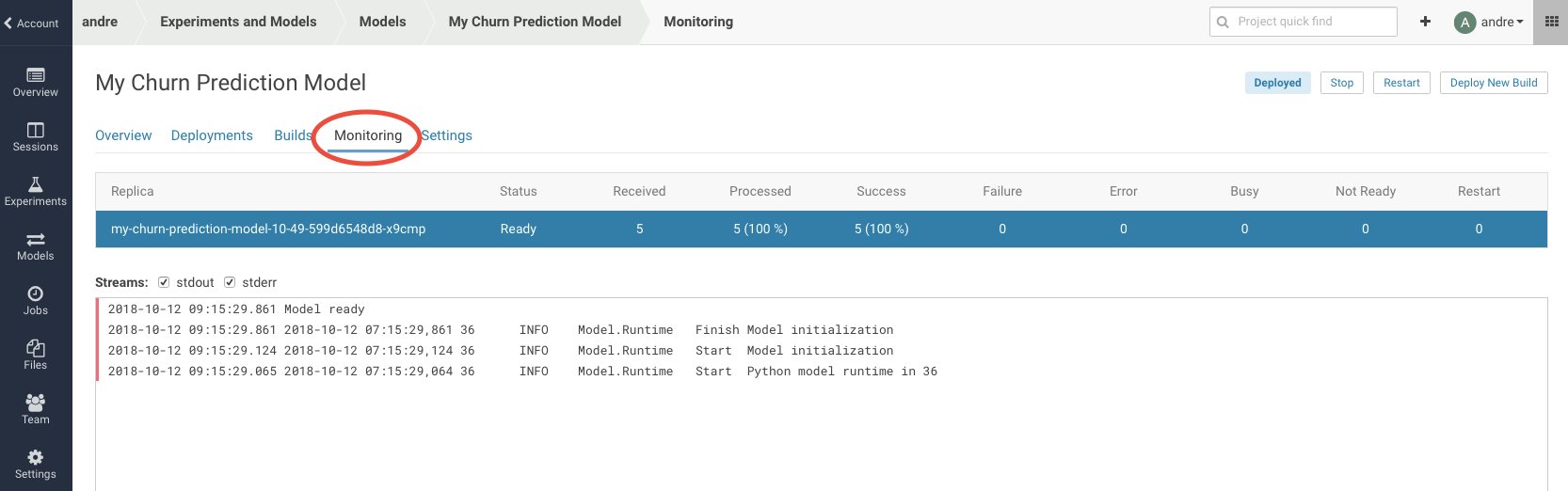


When you get to the re-deployment page, you can increase the number of replica’s.



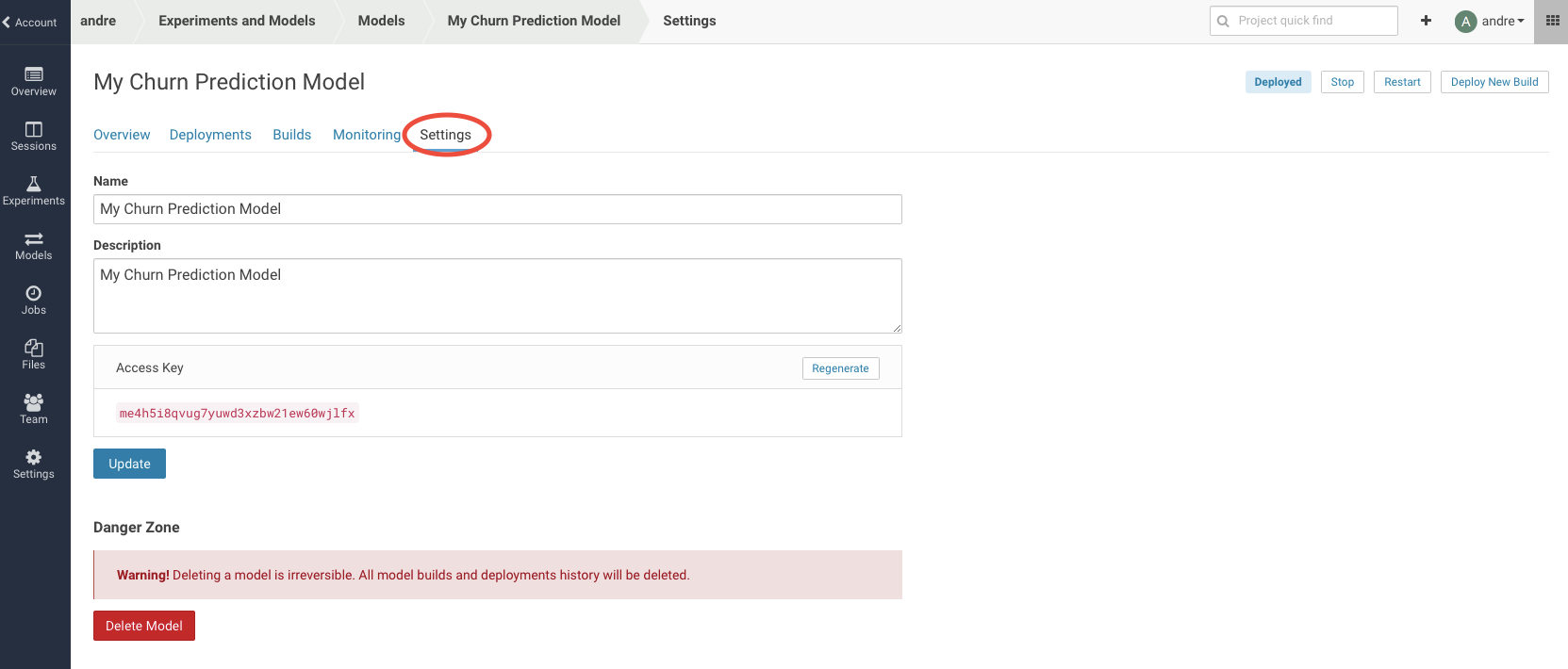
In order not to overload the cluster, hit the ‘Cancel’ button to return to the running model page.

Now, navigate to the ‘Monitoring’ tab.



Several statistics of the model are displayed, like the number of times the model has been called, have been processed, etc.

Logfile information is also available here. The most recent logs are at the top of the pane (see image). stderr logs are displayed next to a red bar while stdout logs are by a green bar. Note that model logs and statistics are only preserved so long as the individual replica is active. When a replica restarts (for example, in case of bad input) the logs also start with a clean slate.  
  
Now, navigate to the ‘Settings’ tab.

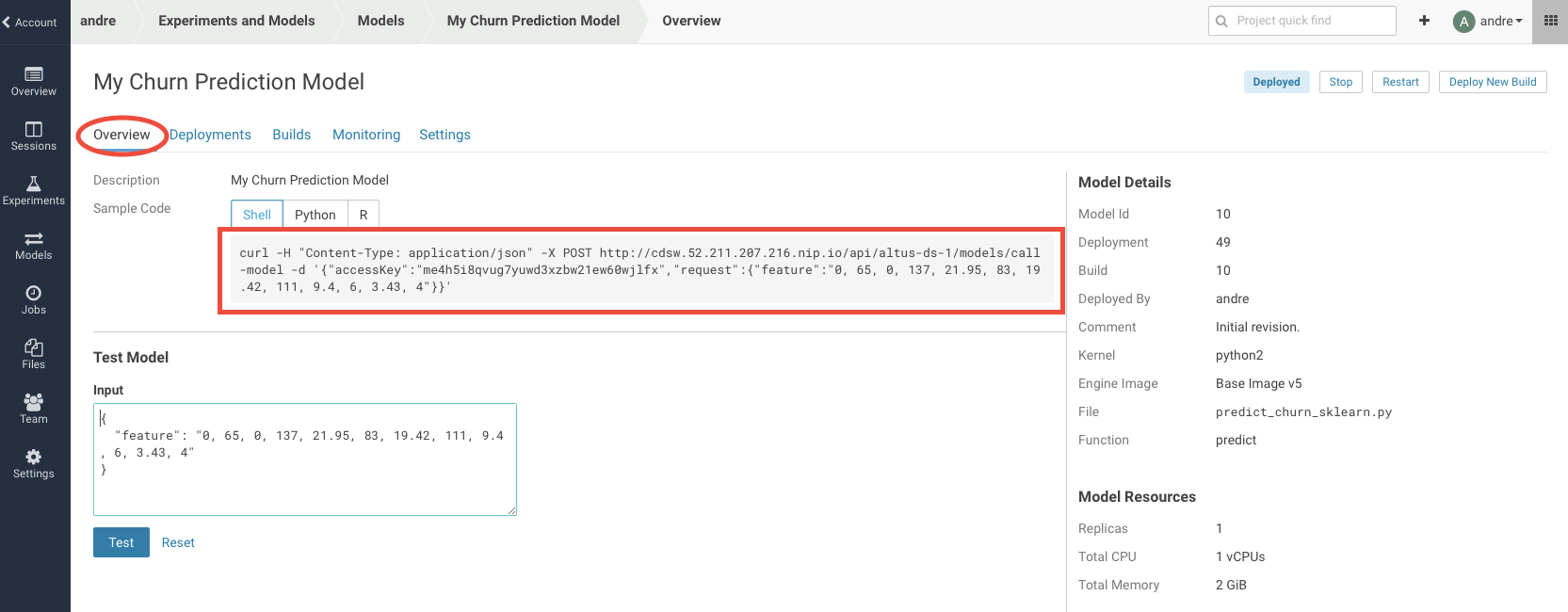


On the settings tab, you will be able to find the “Access Key” that is needed in order to call the model with a REST webservice call.

**Step 5: Test the rest service from a commandline.**

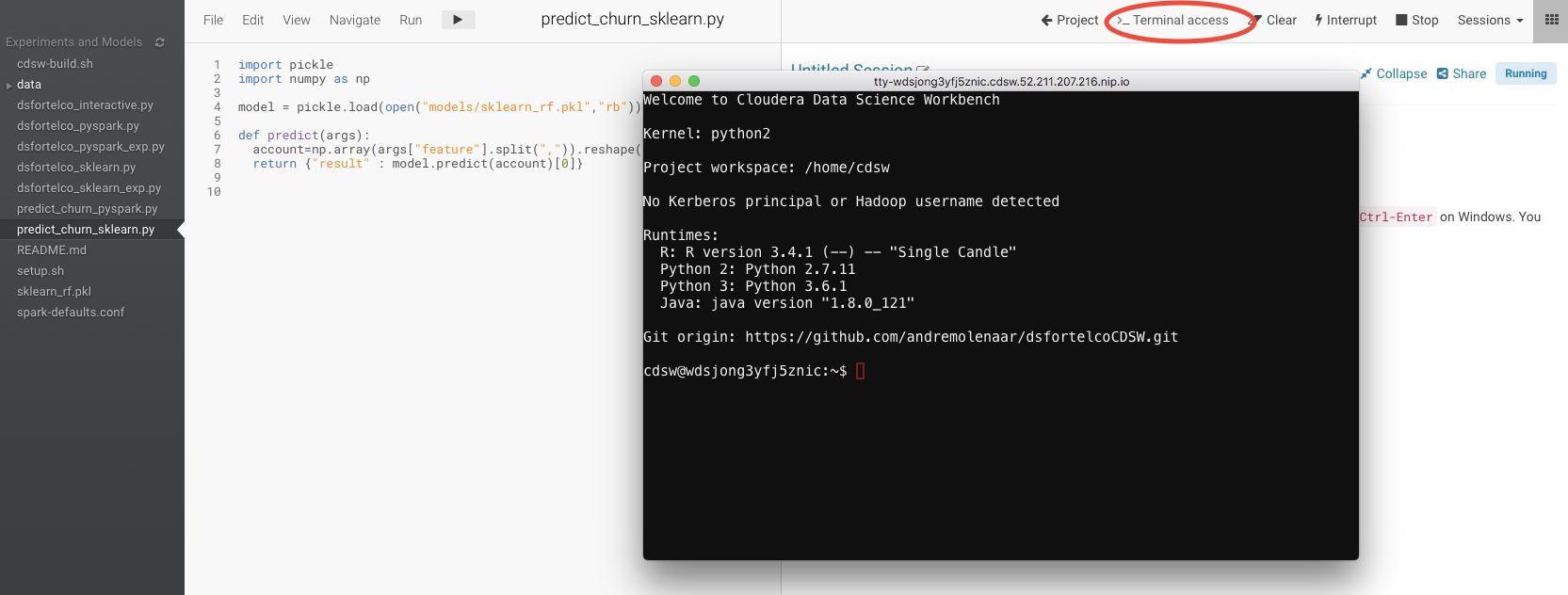
The last step in this workshop, is to test the predict function from another (virtual) machine, using the “curl” tool.

Navigate to the Overview tab of your running model.

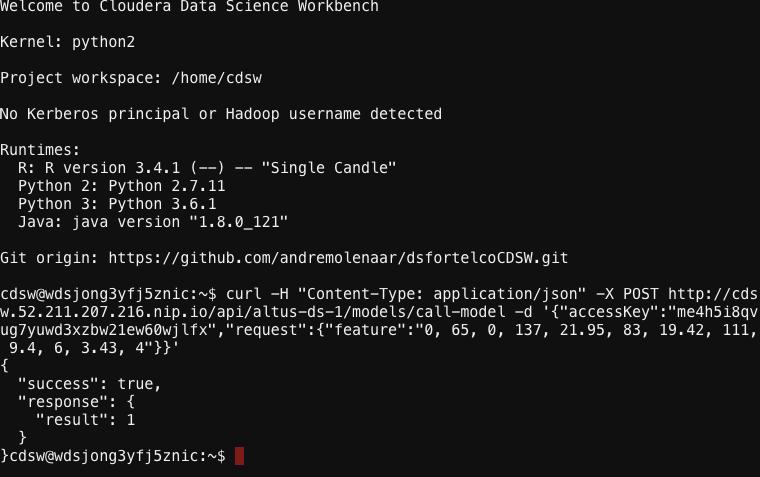


Copy the whole shell statement, starting with ‘curl -H ….”

Open a workbench session, running python 2 with 2 GB of memory. When the session is available, open a Terminal.

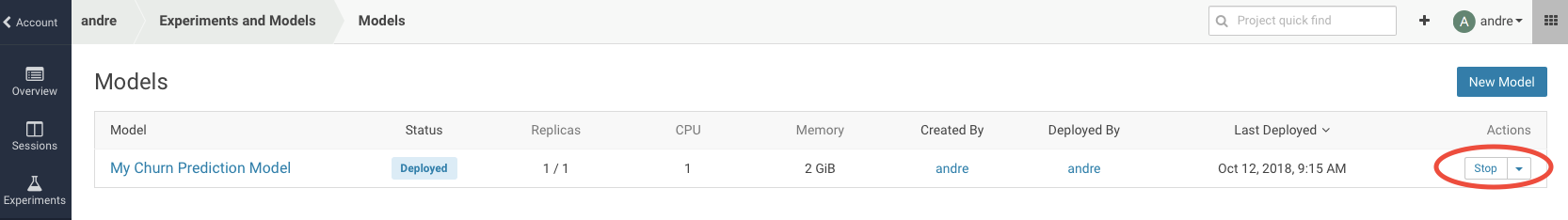


Now, paste the curl statement to the command prompt, and run the statement.



The response shows that the model is still running and making predictions.

That completes our lab with models. Please, freeup some resources for other people and new projects. So stop your workbench session. And from the Models page, also stop your deployed model.



## 