Garmin Workshop

Resources

Web Server: http://34.206.1.88

Registration code: garmin

SSH username: centos

WEB SERVER VM:

==========

Web Server Name	Public DNS Name	Public IP	Private IP	Stoppable
jhorsch-garmin-web	cdp.34.206.1.88.nip.io	34.206.1.88	10.0.1.211	No

CLUSTER VMS:

==========

Cluster Name	Public DNS Name	Public IP	Private IP	Stoppable
jhorsch-garmin-cluster-0	cdp.44.204.245.46.nip.io	44.204.245.46	10.0.1.206	No
jhorsch-garmin-cluster-1	cdp.44.199.252.75.nip.io	44.199.252.75	10.0.1.16	No
jhorsch-garmin-cluster-2	cdp.18.209.214.123.nip.io	18.209.214.123	10.0.1.96	No
jhorsch-garmin-cluster-3	cdp.3.237.182.114.nip.io	3.237.182.114	10.0.1.30	No
jhorsch-garmin-cluster-4	cdp.18.209.210.179.nip.io	18.209.210.179	10.0.1.202	No
jhorsch-garmin-cluster-5	cdp.44.193.24.187.nip.io	44.193.24.187	10.0.1.177	No

Health checks:

instance	ip address	WEB	CM	EFM	NIFI	NREG	SREG	SMM	HUE	SSB	CDSW	Model Status	Viz Status
aws_instance.web	34.206.1.88	Ok											
aws_instance.cluster[0]	44.204.245.46		Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok			
aws_instance.cluster[1]	44.199.252.75		Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok			
aws_instance.cluster[2]	18.209.214.123		Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok			
aws_instance.cluster[3]	3.237.182.114		Ok	Ok	Ok	Ok	Ok	Ok		Ok			
aws_instance.cluster[4]	18.209.210.179		Ok	Ok		Ok	Ok	Ok		Ok			
aws_instance.cluster[5]	44.193.24.187		Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok			

CML/CDSW Labs

Experiments and Models

Labs summary

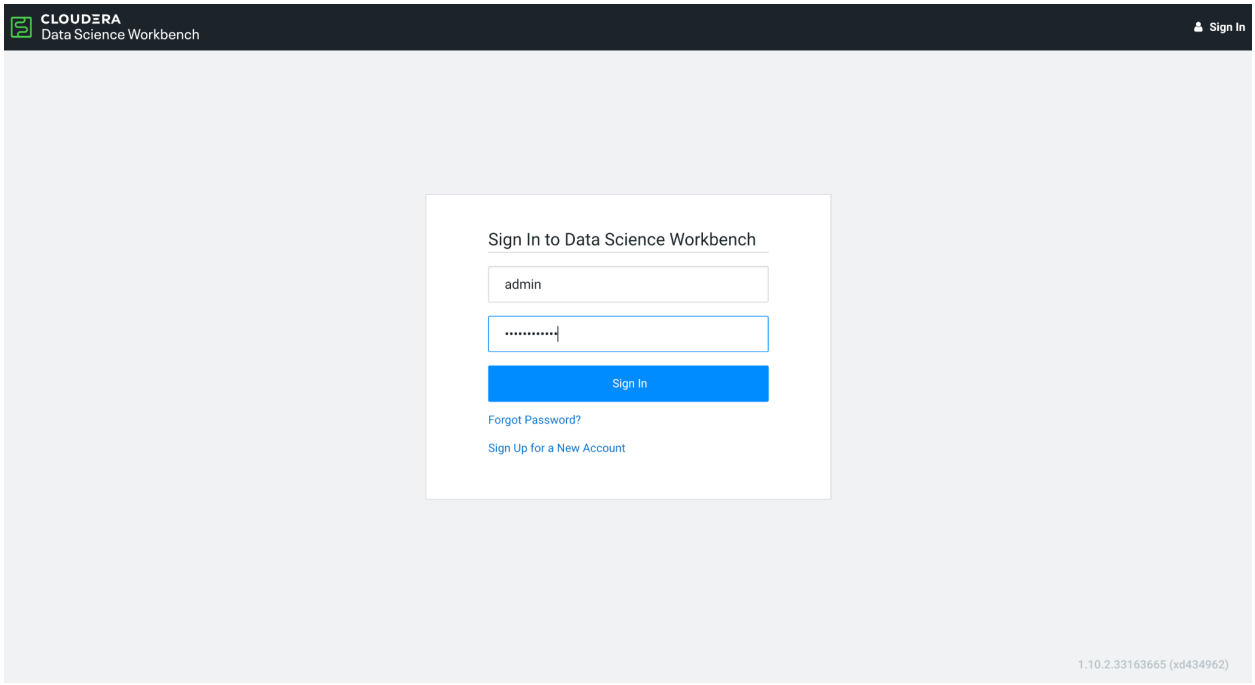
- Lab 1 - Train the model.
- Lab 2 - Deploy the Model.

Lab 1 - CDSW: Train the model

In this and the following lab, you will wear the hat of a Data Scientist. You will write the model code, train it several times and finally deploy the model to Production.

STEP 1: Configure CDSW

1. Open CDSW Web UI and log in as admin/Supersecret1, if you haven't yet done so.



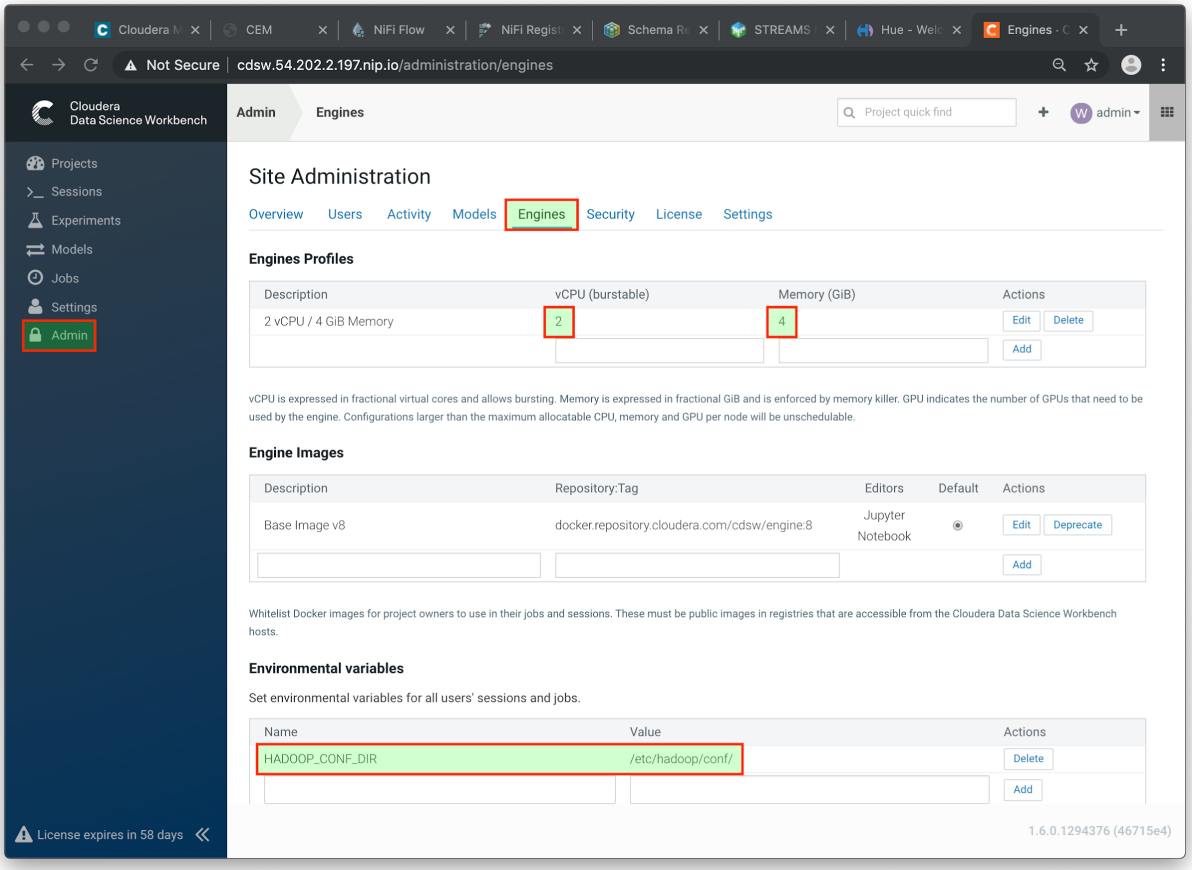
2. Navigate to the CDSW Admin page to fine tune the environment:
    i. In the Engines tab, add in Engines Profiles a new engine (docker image) with 2 vCPUs and 4 GB RAM, while deleting the default engine.
    ii. Check if the following variable already exists under Environmental Variables. If not, add it:

```
Unset
```

    iii.

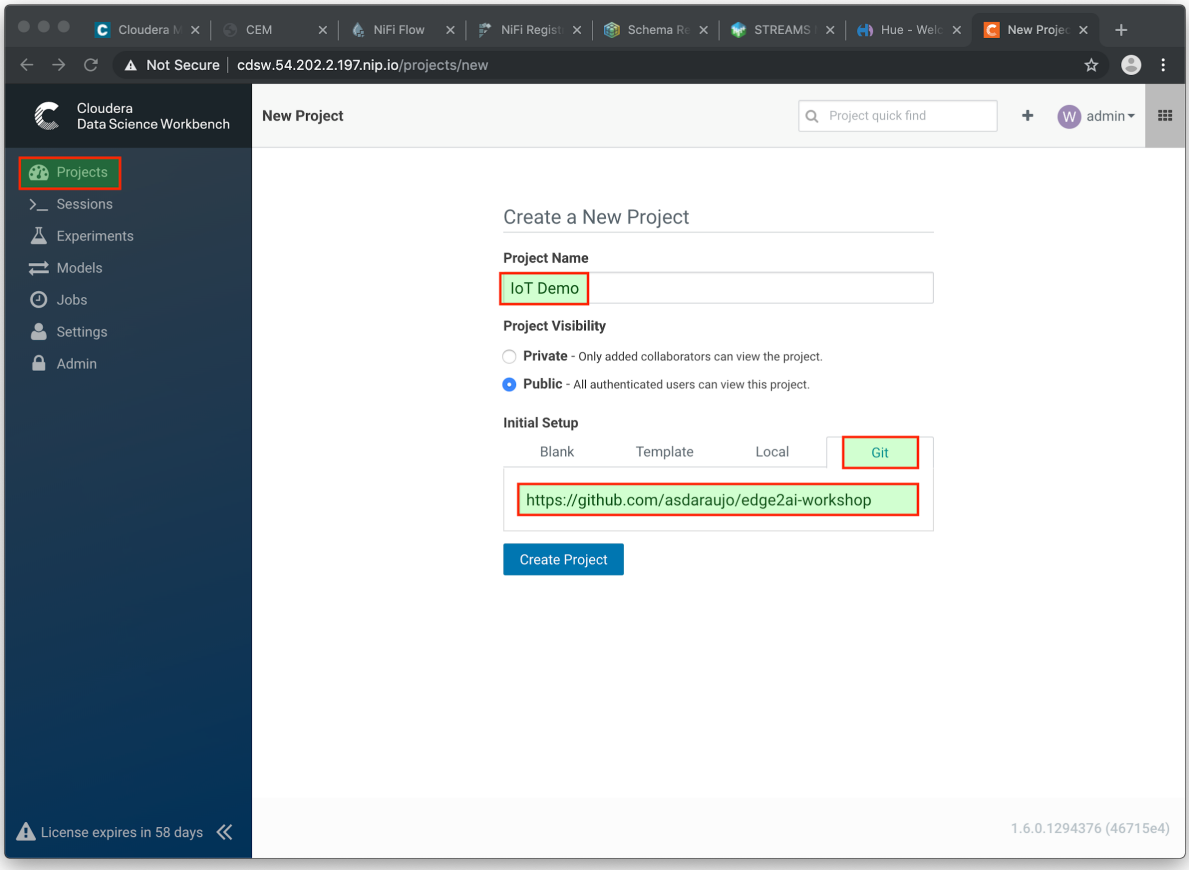
```
HADOOP_CONF_DIR=/etc/hadoop/conf/
```

iv.

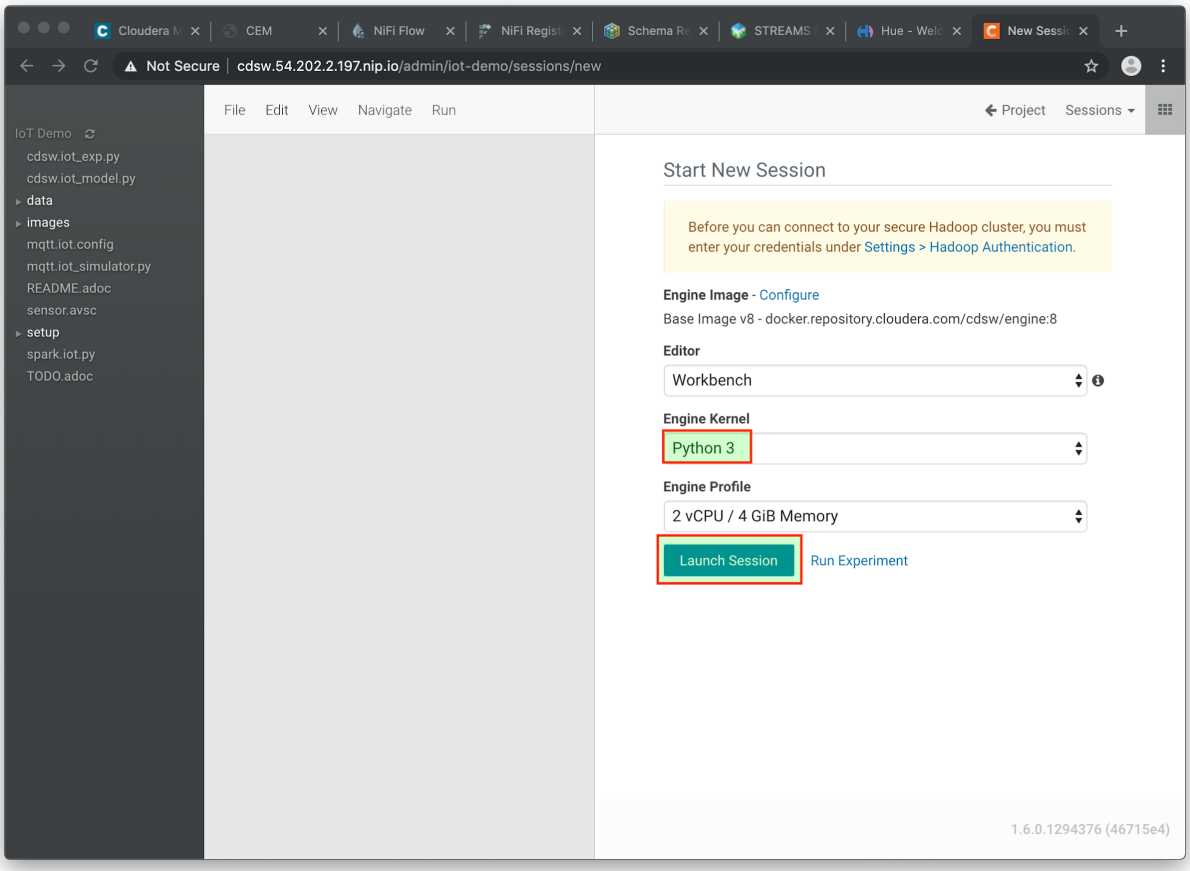

Not Secure | cdsw.54.202.2.197.nip.io/administration/engines
Cloudera Data Science Workbench
Admin
Engines
Project quick find
admin
Projects
Sessions
Experiments
Models
Jobs
Settings
Admin
Site Administration
Overview
Users
Activity
Models
Engines
Security
License
Settings
Engines Profiles
Description
vCPU (burstable)
Memory (GiB)
Actions
2 vCPU / 4 GiB Memory
2
4
Edit
Delete
Add
vCPU is expressed in fractional virtual cores and allows bursting. Memory is expressed in fractional GiB and is enforced by memory killer. GPU indicates the number of GPUs that need to be used by the engine. Configurations larger than the maximum allocatable CPU, memory and GPU per node will be unschedulable.
Engine Images
Description
Repository:Tag
Editors
Default
Actions
Base Image v8
docker.repository.cloudera.com/cdsw/engine:8
Jupyter Notebook
Edit
Deprecate
Add
Whitelist Docker images for project owners to use in their jobs and sessions. These must be public images in registries that are accessible from the Cloudera Data Science Workbench hosts.
Environmental variables
Set environmental variables for all users' sessions and jobs.
Name
Value
Actions
HADOOP_CONF_DIR
/etc/hadoop/conf/
Delete
Add
License expires in 58 days
1.6.0.1294376 (46715e4)

STEP 2: Create the project

1. Return to the main page and click on New Project, using this GitHub project as the source: https://github.com/cloudera-labs/edge2ai-workshop



2. Now that your project has been created, click on Open Workbench and start a Python3 session:



3. Once the Engine is ready, run the following command to install some required libraries:

```
Unset
```

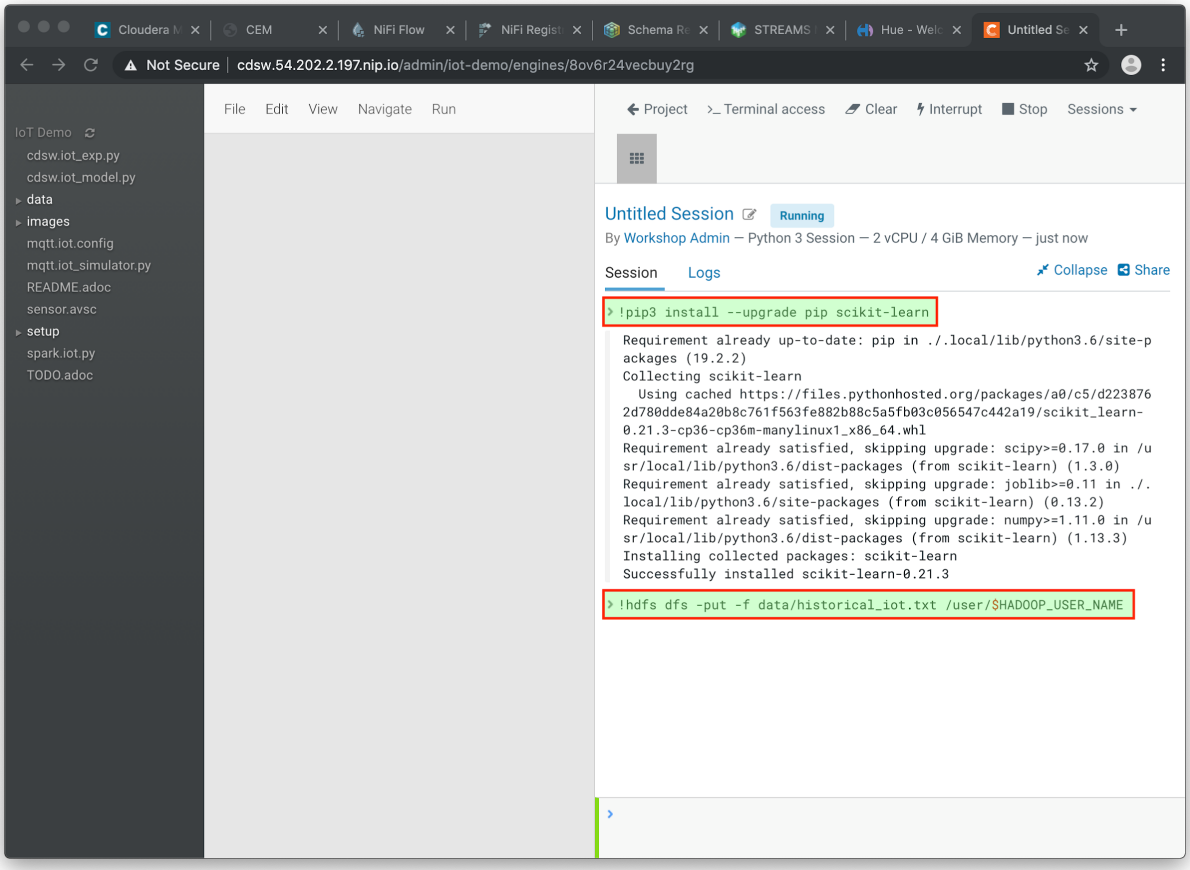
4.
```
!pip3 install --upgrade pip scikit-learn
```

5. The project comes with a historical dataset. Copy this dataset into HDFS:

```
Unset
```

6.
```
!hdfs dfs -put -f data/historical_iot.txt /user/$HADOOP_USER_NAME
```

7.



8. You're now ready to run the Experiment to train the model on your historical data.
9. You can stop the Engine at this point.

STEP 3: Examine cdsw.iot_exp.py

Open the file cdsw.iot_exp.py. This is a python program that builds a model to predict machine failure (the likelihood that this machine is going to fail). There is a dataset available on hdfs with customer data, including a failure indicator field.
The program is going to build a failure 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:

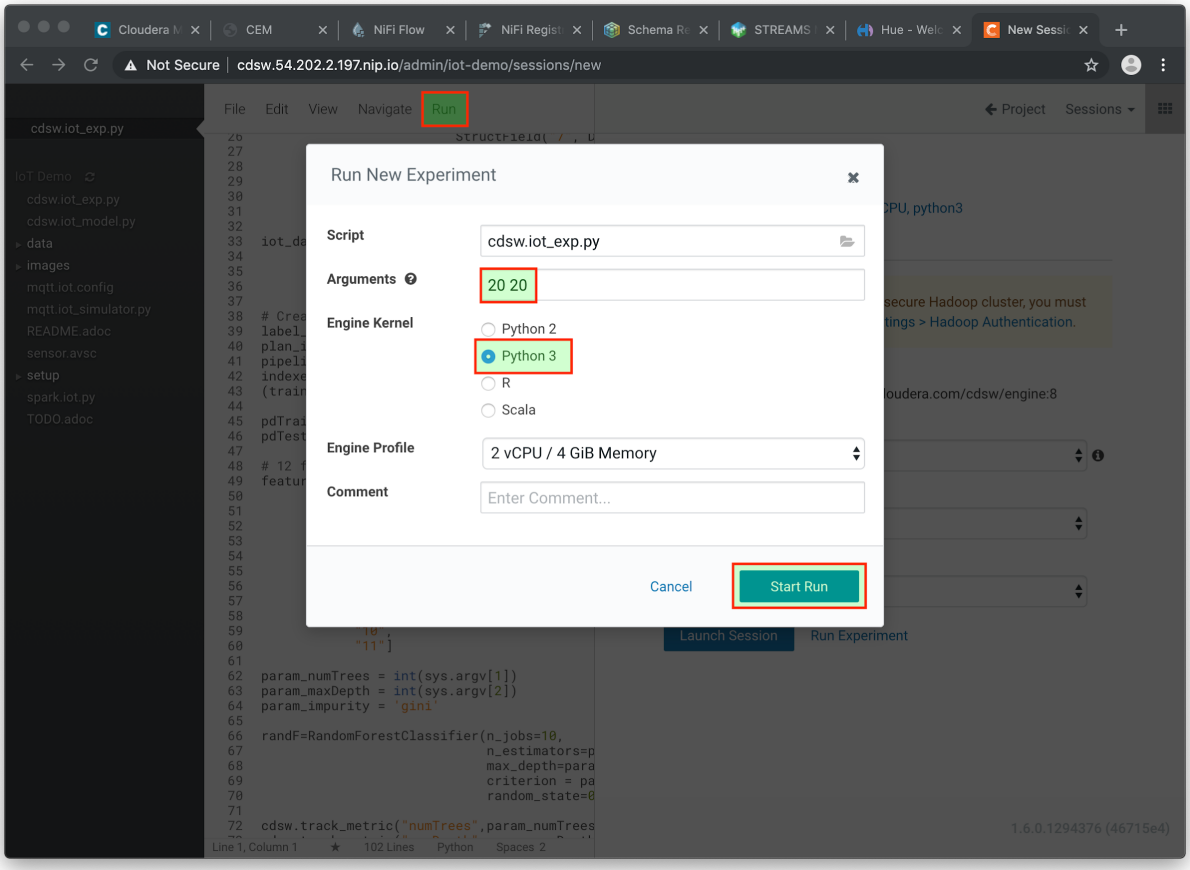
- numTrees: Number of trees in the forest.
  Increasing the number of trees will decrease the variance in predictions, improving the model's test-time accuracy. Training time increases roughly linearly in the number of trees.
- maxDepth: Maximum depth of each tree in the forest.
  Increasing the depth 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 cdsw.iot_exp.py program, these parameters can be passed to the program at runtime, to these python variables:
param_numTrees = int(sys.argv[1])
param_maxDepth = int(sys.argv[2])
Also note 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 4: Run the experiment for the first time

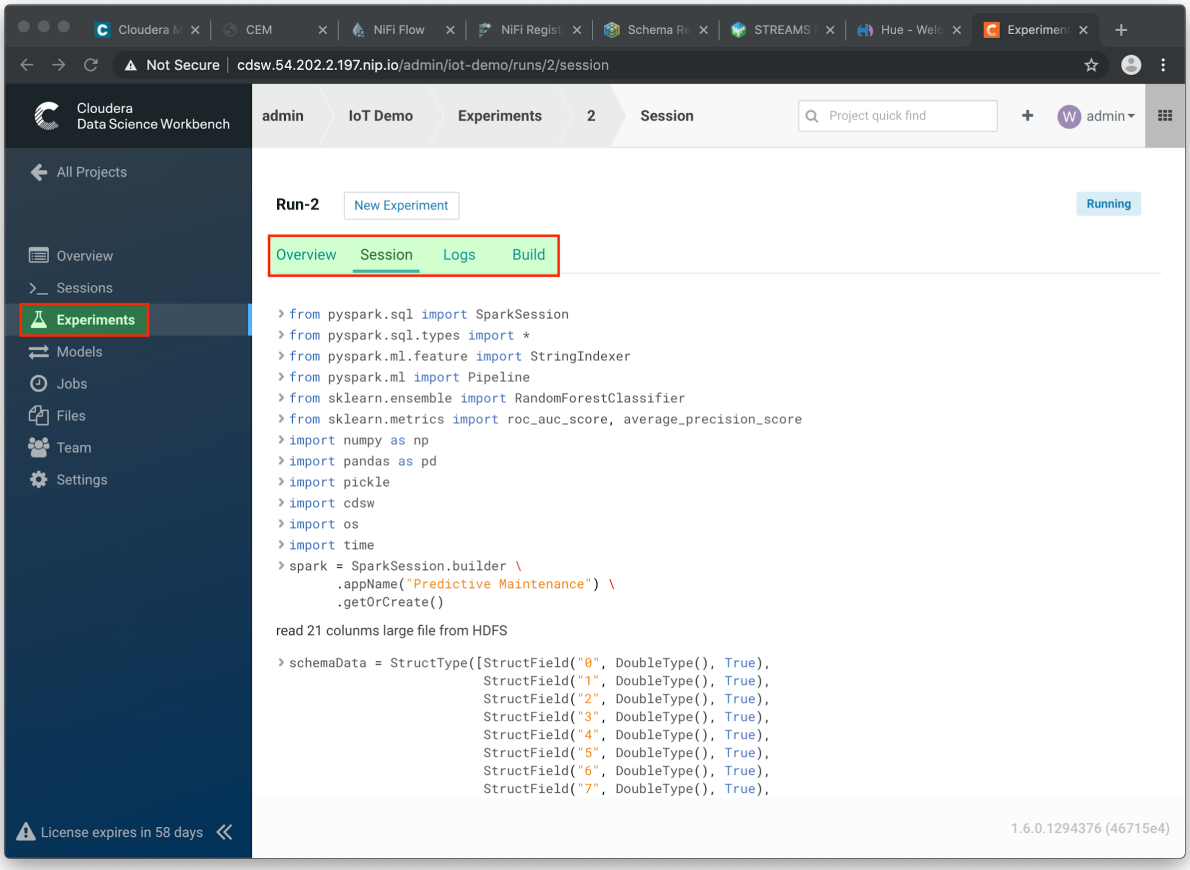
1. Now, run the experiment using the following parameters:
   numTrees = 20 numDepth = 20
2. From the menu, select Run → Run Experiments…. Now, in the background, the Data Science Workbench environment will spin up a new docker container, where this

program will run.



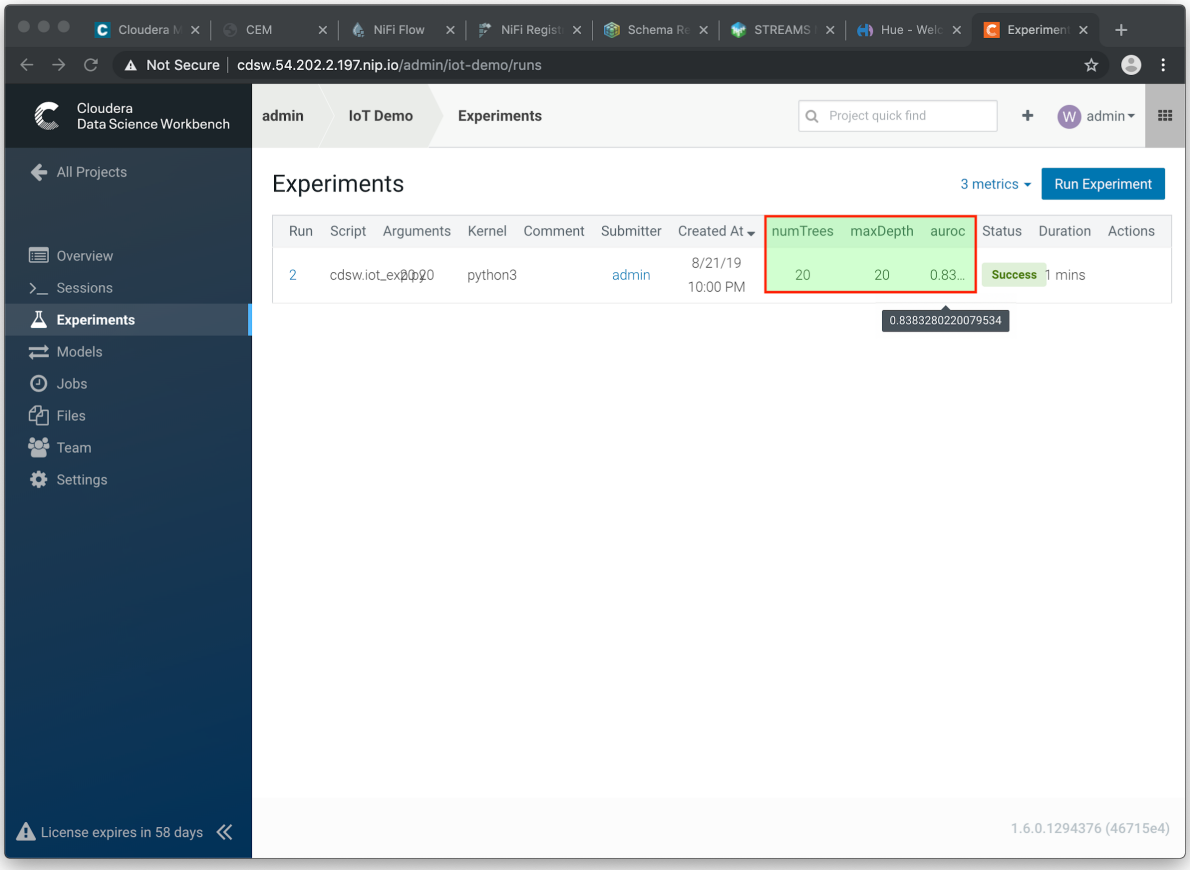
3. Go back to the Projects page in CDSW, and hit the Experiments button.
4. If the Status indicates Running, you have to wait till the run is completed. In case the status is Build Failed or Failed , check the log information. This is accessible by clicking on the run number of your experiments. There you can find the session log, as well as

the build information.



5. 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.



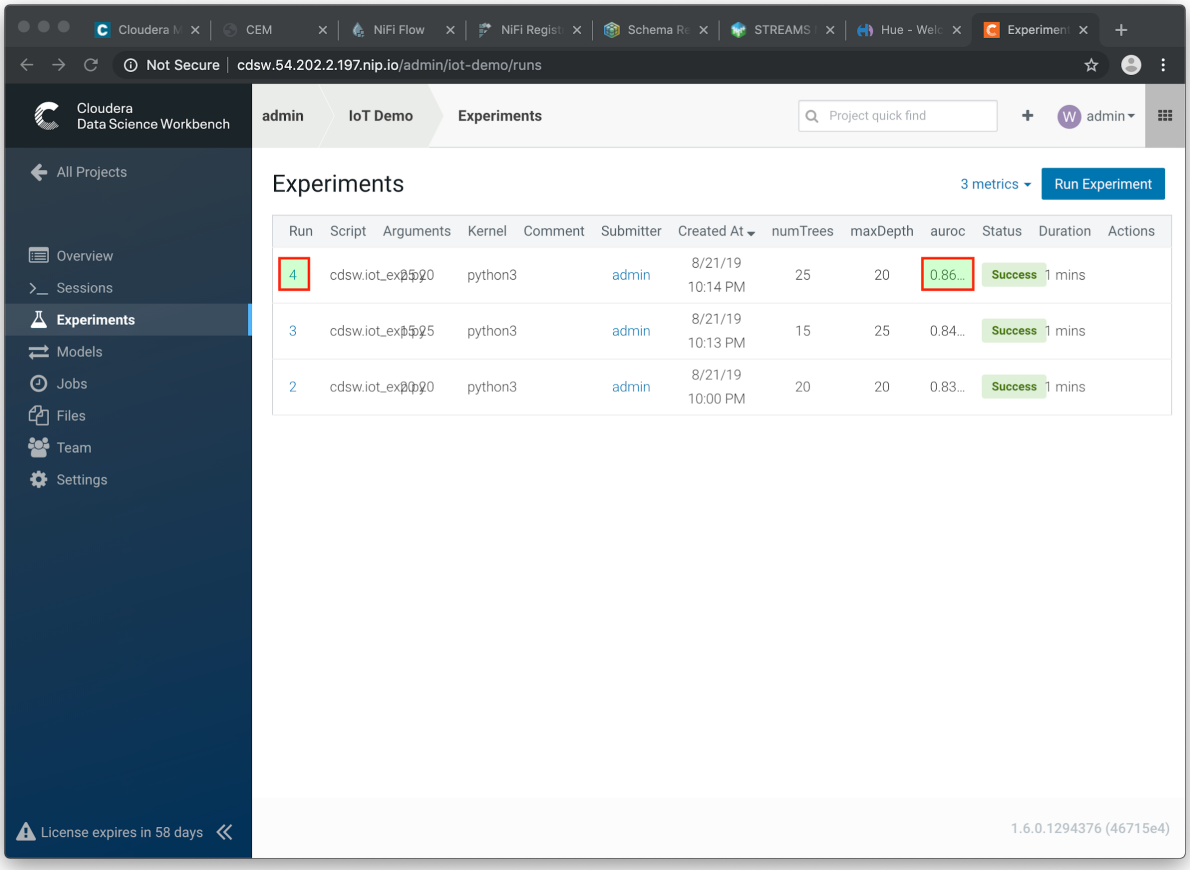
6. In this example, ~0.8383. Not bad, but maybe there are better hyper parameter values available.

STEP 5: Re-run the experiment several times

Go back to the Workbench and run the experiment 2 more times and try different values for NumTrees and NumDepth. Try the following values:
NumTrees NumDepth
15 25

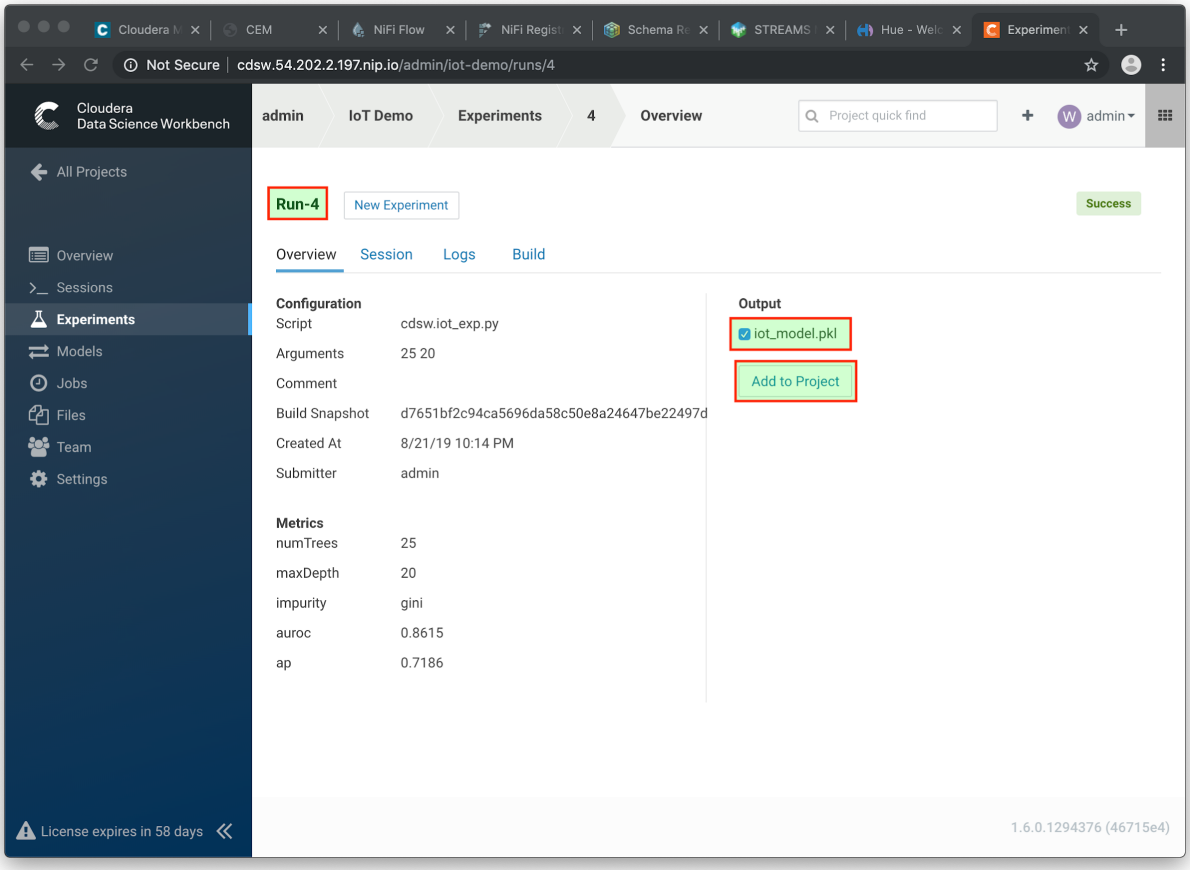
1. 25 20
2. When all runs have completed successfully, check which parameters had the best quality (best predictive value). This is represented by the highest area under the curve:

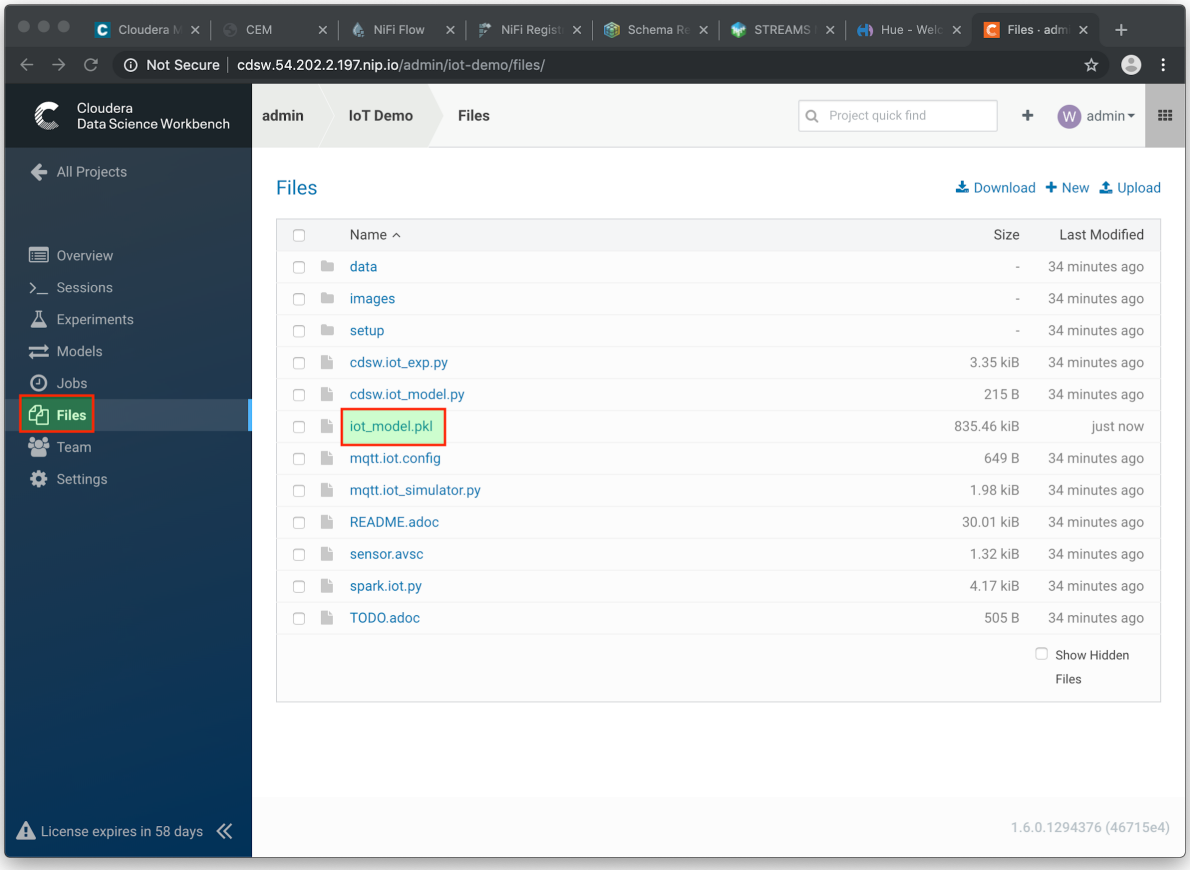
auroc metric.



STEP 6: Save the best model to your environment

1. Select the run number with the best predictive value (in the example above, experiment 4).
2. In the Overview screen of the experiment, you can see that the model, in Pickle format (.pkl), is captured in the file iot_model.pkl. Select this file and hit the Add to Project button. This will copy the model to your project directory.


Not Secure
cdsw.54.202.2.197.nip.io/admin/iot-demo/runs/4
Cloudera
Data Science Workbench
admin
IoT Demo
Experiments
4
Overview
Project quick find
admin
All Projects
Overview
Sessions
Experiments
Models
Jobs
Files
Team
Settings
Run-4
New Experiment
Success
Overview
Session
Logs
Build
Configuration
Script
cdsw.iot_exp.py
Arguments
25 20
Comment
Build Snapshot
d7651bf2c94ca5696da58c50e8a24647be22497d
Created At
8/21/19 10:14 PM
Submitter
admin
Metrics
numTrees
25
maxDepth
20
impurity
gini
auroc
0.8615
ap
0.7186
Output
iot_model.pkl
Add to Project
License expires in 58 days
1.6.0.1294376 (46715e4)

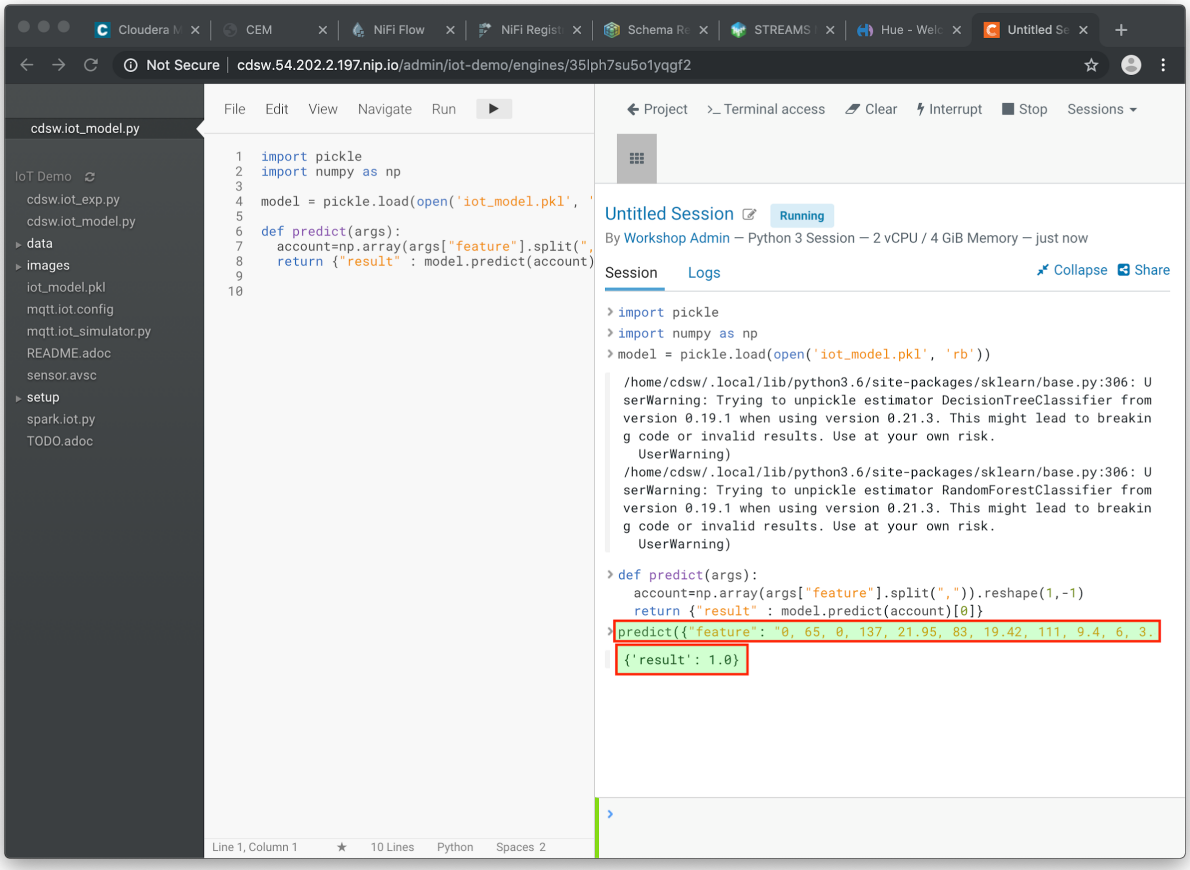


Lab 2 - CDSW: Deploy the model

STEP 1: Examine the program cdsw.iot_model.py

1. Open the project you created in the previous lab and examine the file in the Workbench. This PySpark program uses the pickle.load mechanism to deploy models. The model is loaded from the iot_modelf.pkl file, which was saved in the previous lab from the experiment with the best predictive model.
   There program also contains the predict definition, which is the function that calls the model, passing the features as parameters, and will return a result variable.
2. Before deploying the model, try it out in the Workbench: launch a Python3 engine and run the code in file cdsw.iot_model.py. Then call the predict() method from the prompt:

predict({"feature": "0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4"})



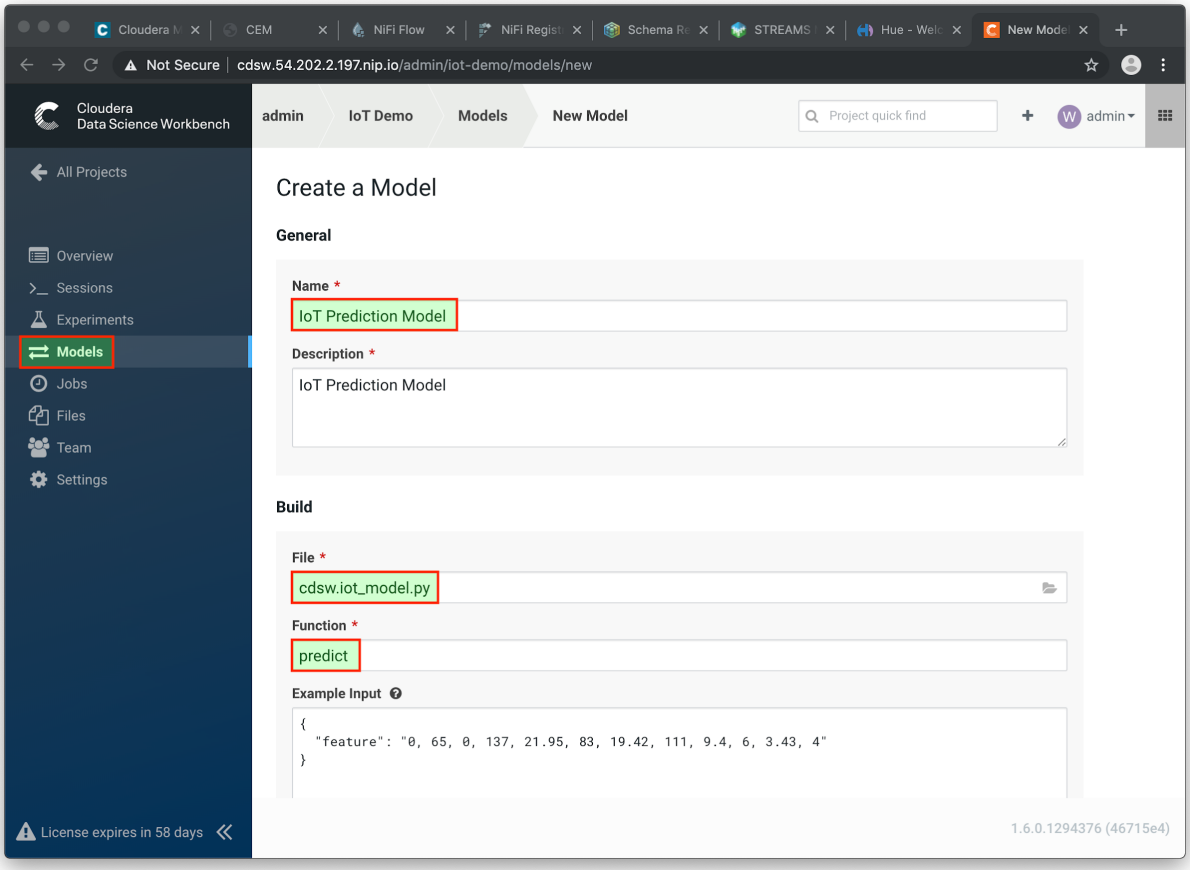
3. The functions returns successfully, so we know we can now deploy the model. You can now stop the engine.

STEP 2: Deploy the model

From the main page of your project, select the Models button. Select New Model and specify the following configuration:
Name: IoT Prediction Model
Description: IoT Prediction Model
File: cdsw.iot_model.py
Function: predict
Example Input: {"feature": "0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4"}

Kernel: Python 3
Engine: 2 vCPU / 4 GB Memory

1. Replicas: 1

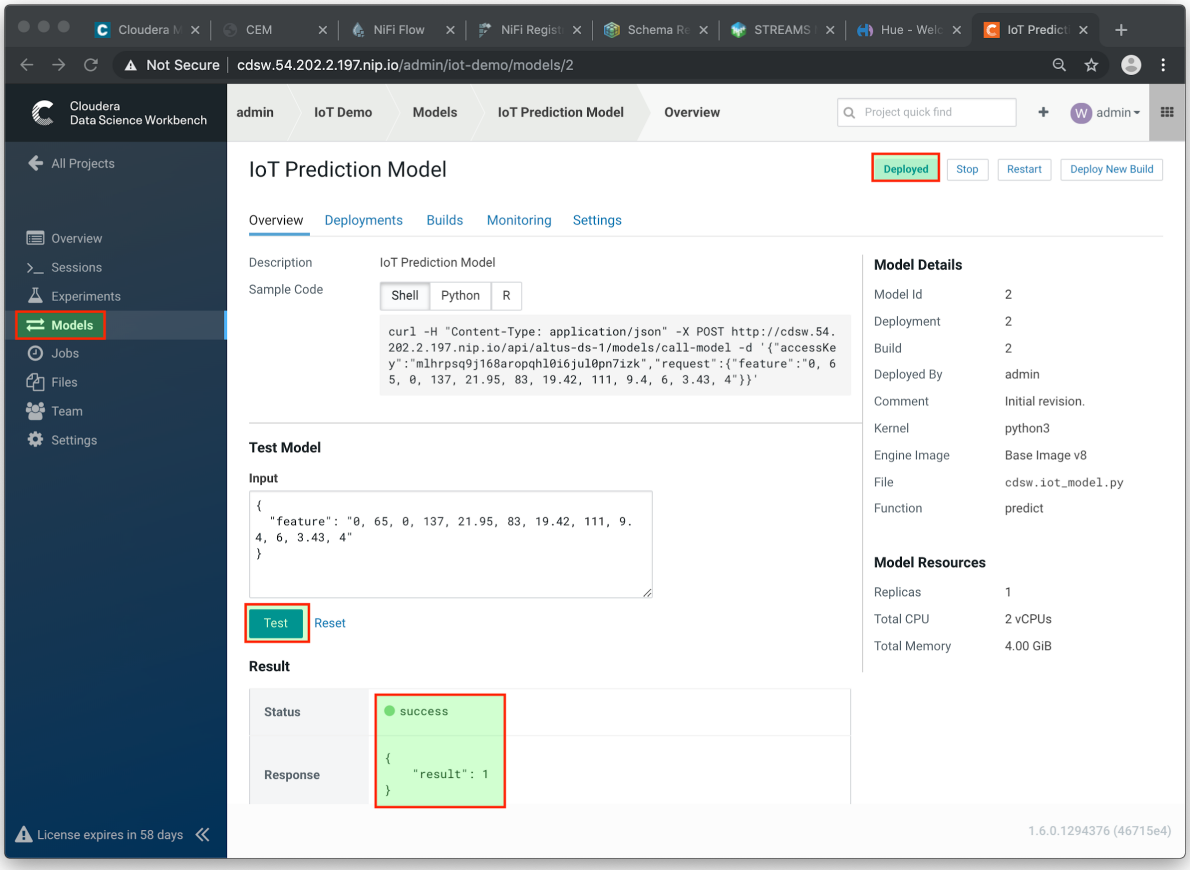


2. After all parameters are set, click on the Deploy Model button. Wait till the model is deployed. This can take several minutes.

STEP 3: Test the deployed model

1. When your model status change to Deployed, click on the model name link to go to the Model's Overview page. From the that page, click on the Test button to check if the model is working.
2. The green circle with the success status indicates that our REST call to the model is working. The 1 in the response {"result": 1}, means that the machine from where these

temperature readings were collected is unlikely to experience a failure.



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"

3. }
4. With these input parameters, the model returns 0, which means that the machine is likely to break.

Creating Dashboards with Cloudera Data Viz

In this workshop you will create a simple interactive real-time dashboard to visualize sensor data that is being stored in Kudu.

The data you will use is the sensor data collected and processed in previous workshops (see Preparation below).

Preparation

This workshop builds upon the content developed in the Edge and Nifi workshops.
To clean your environment and make it ready for the beginning of this lab, please SSH to your cluster host and run the following command:

Note	The command below will undo everything done in the cluster in previous workshops.

/tmp/resources/reset-to-lab.sh dataviz 1

SSH to your host as centos/supersecret1 and run the above command

Labs summary

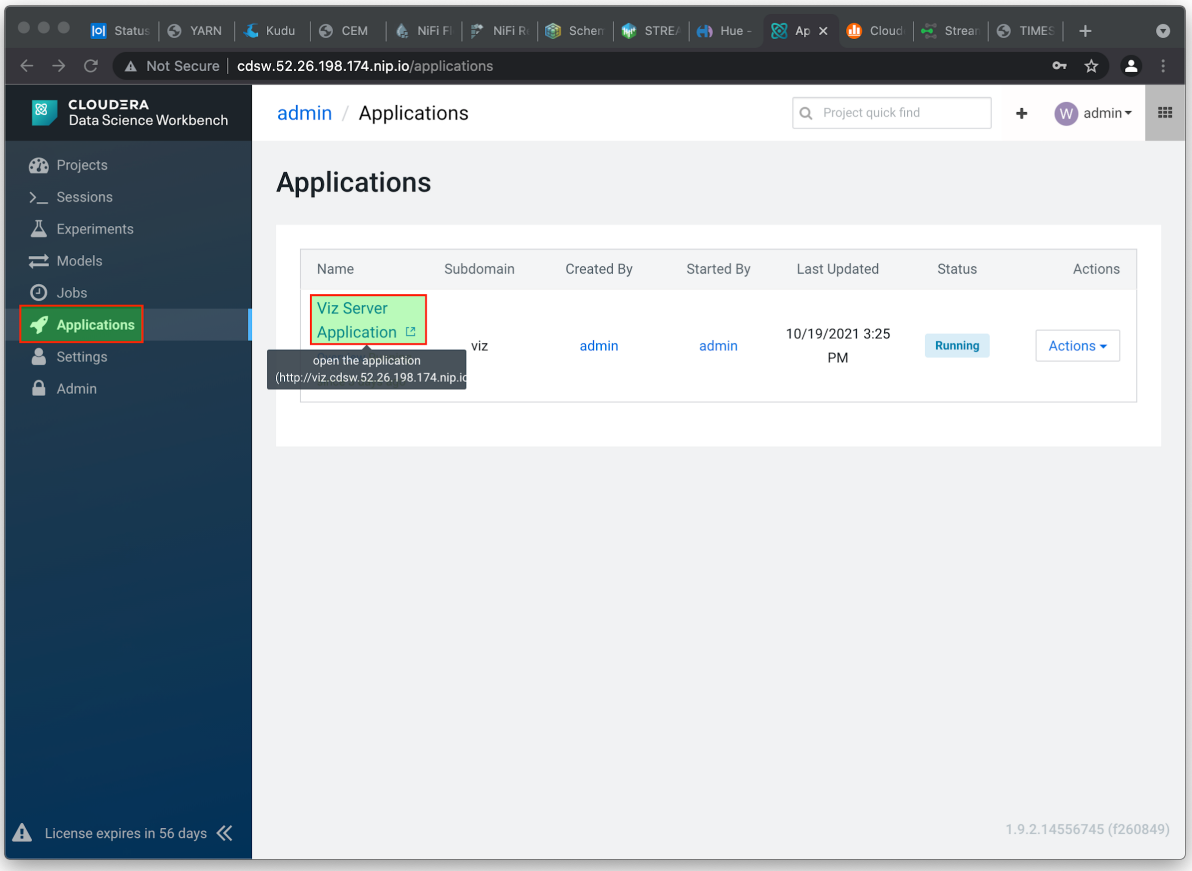
- Lab 1 - Navigate to Cloudera Data Visualization
- Lab 2 - Creating a new connection
- Lab 3 - Exploring the data
- Lab 4 - Creating a dashboard
- Lab 5 - Adding a chart

Lab 1 - Navigate to Cloudera Data Visualization

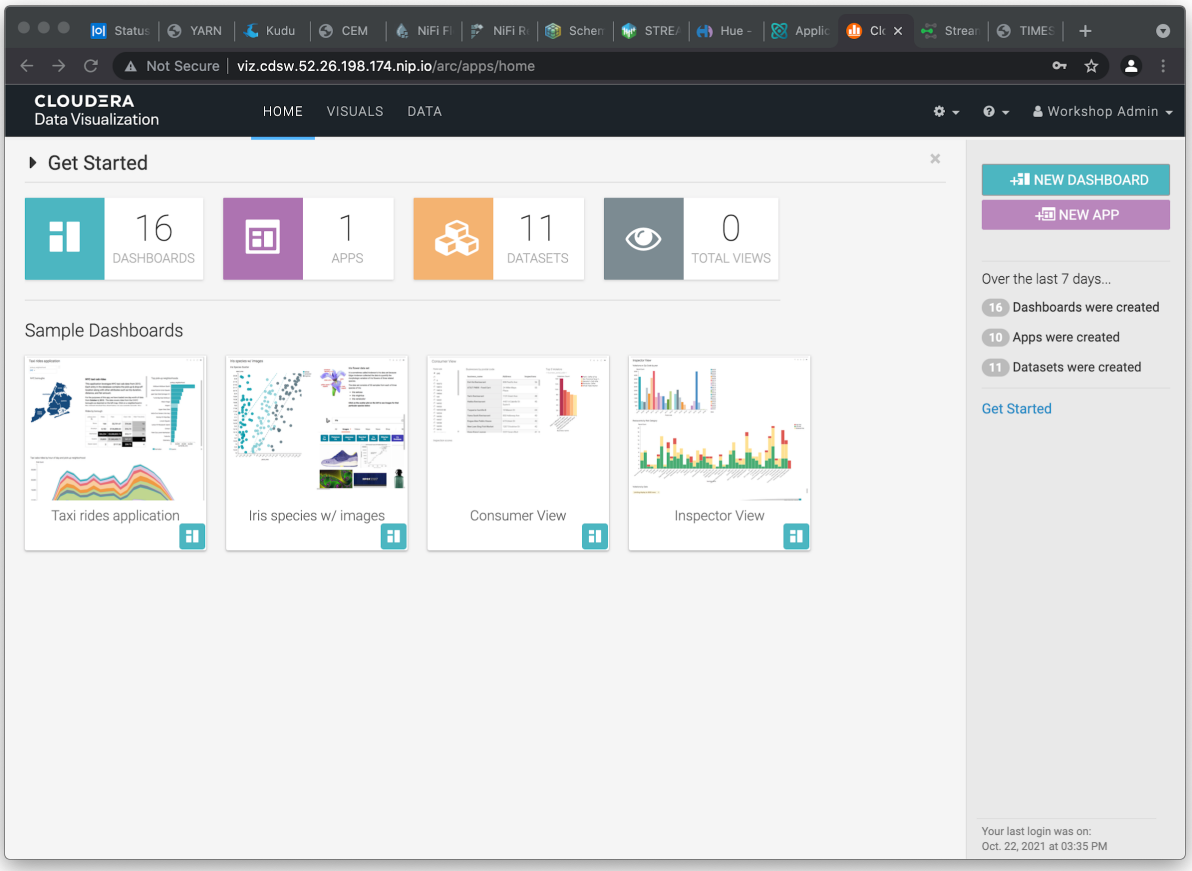
This lab shows you how to navigate to Cloudera Data Visualization (DataViz) page.
If you are in a guided workshop you may already been given the link to the DataViz page. If that's the case, feel free to skip to the next lab.

1. Open CDP Data Visualization and log in
   CDP Data Visualization can be accessed through the Cloudera Data Science Workbench (CDSW). Follow the navigation steps below if you don't know how to get there:
   i. In Cloudera Manager, click on Clusters > Cloudera Data Science Workbench.
   ii. On the CDSW page, click on the CDSW Web UI link.
   iii. Log on to CDSW.

iv. On the CDSW page, click on Applications and then on the "Viz Server Application", which has been previously set up for the workshop.



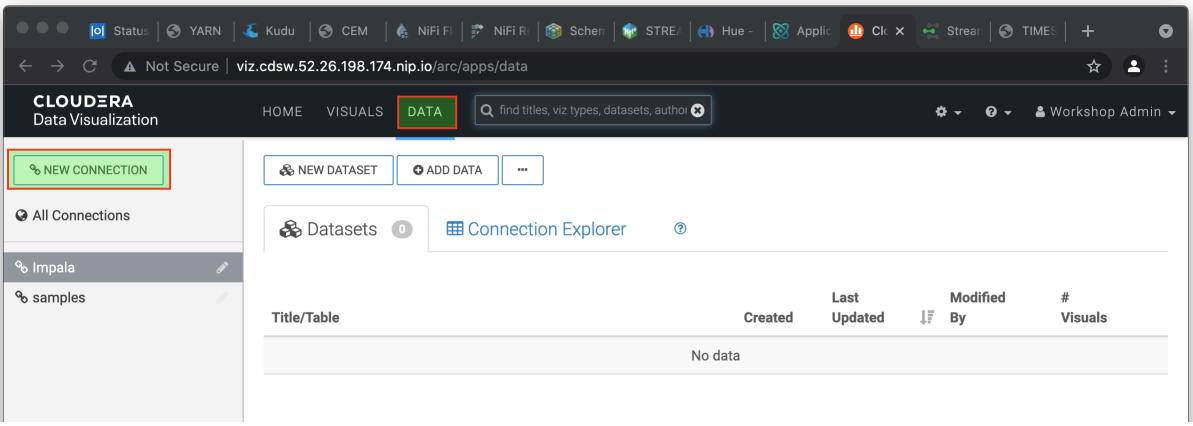
v. Log in to the Cloudera Data Visualization application. After logging in you should see the application home page:



Lab 2 - Creating a new connection

Kudu is purely a storage engine and does not provide a SQL interface for querying. SQL access to Kudu is done through an Impala engine, which is what you will use in this workshop. You will set up a new connection to the Impala engine to use for your dashboard queries.

1. Select the Data tab and click on NEW CONNECTION.

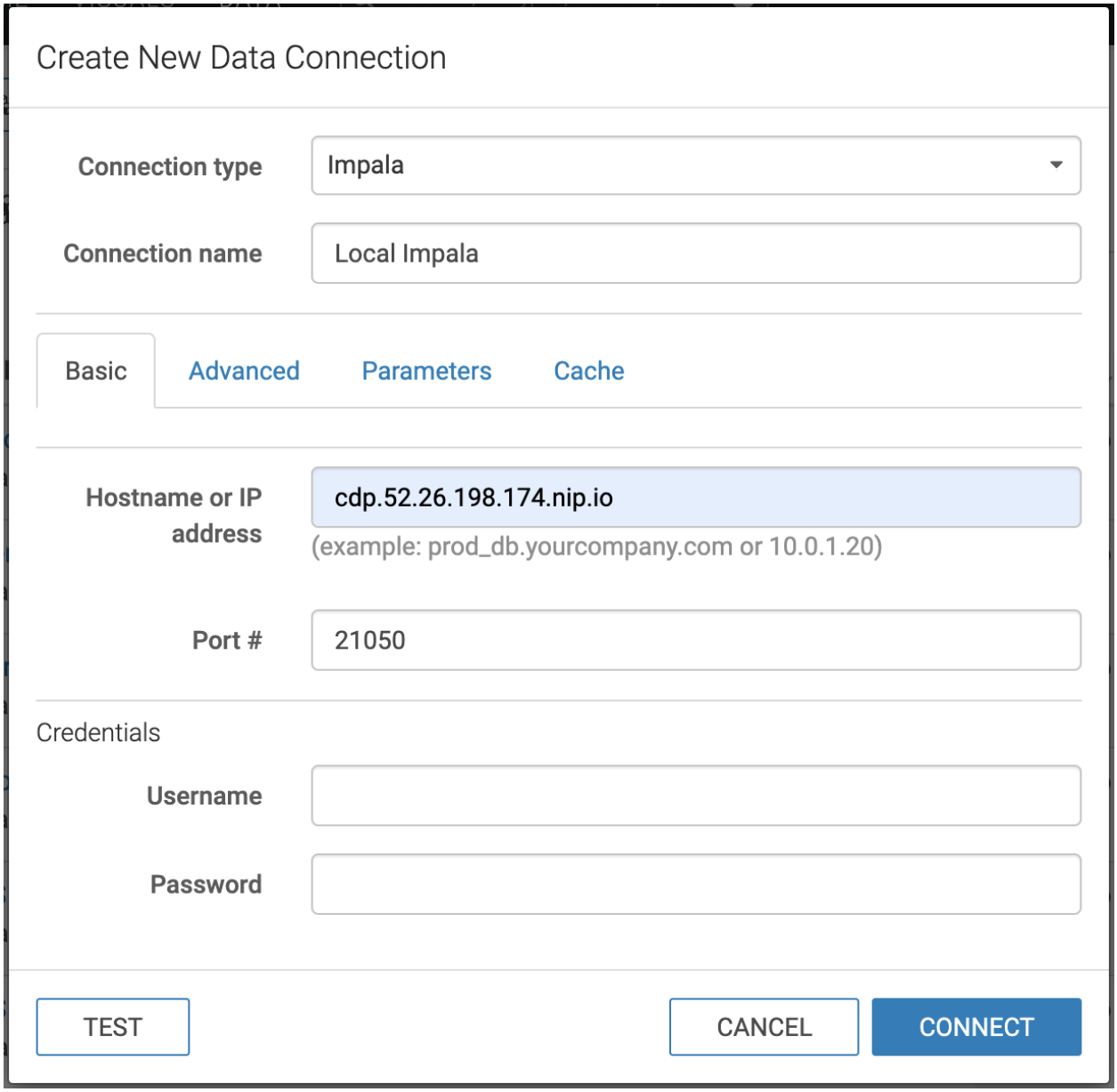


At the top of the form, set the following properties:
Connection type: Impala

2. Connection name: Local Impala

In the Basic tab set the following:
Hostname: <CLUSTER_HOSTNAME> (something like: cdp.x.x.x.x.nip.io)
Port #: 21050
Username: [leave blank]

3. Password: [leave blank]



In the Advanced tab set the following:
Connection mode: Binary
Socket type: Normal

4. Authentication mode: NoSasl

5. Click on TEST to test the connection.
   You should see "Connection Verified", as shown below.

Create New Data Connection

Connection type: Impala

Connection name: Local Impala

Basic | Advanced | Parameters | Cache

Connection mode: (•) Binary ( ) HTTP

Socket type: (•) Normal ( ) SSL ( ) SSL with certificate

Authentication mode: (•) NoSasl ( ) Plain ( ) LDAP ( ) Kerberos

Query Timeout: 60

Session Timeout: 60

Socket Timeout: 60

Queue Depth:

Impersonation: [ ] Enabled

Trusted Impersonation: [ ] Enabled

Connection Verified

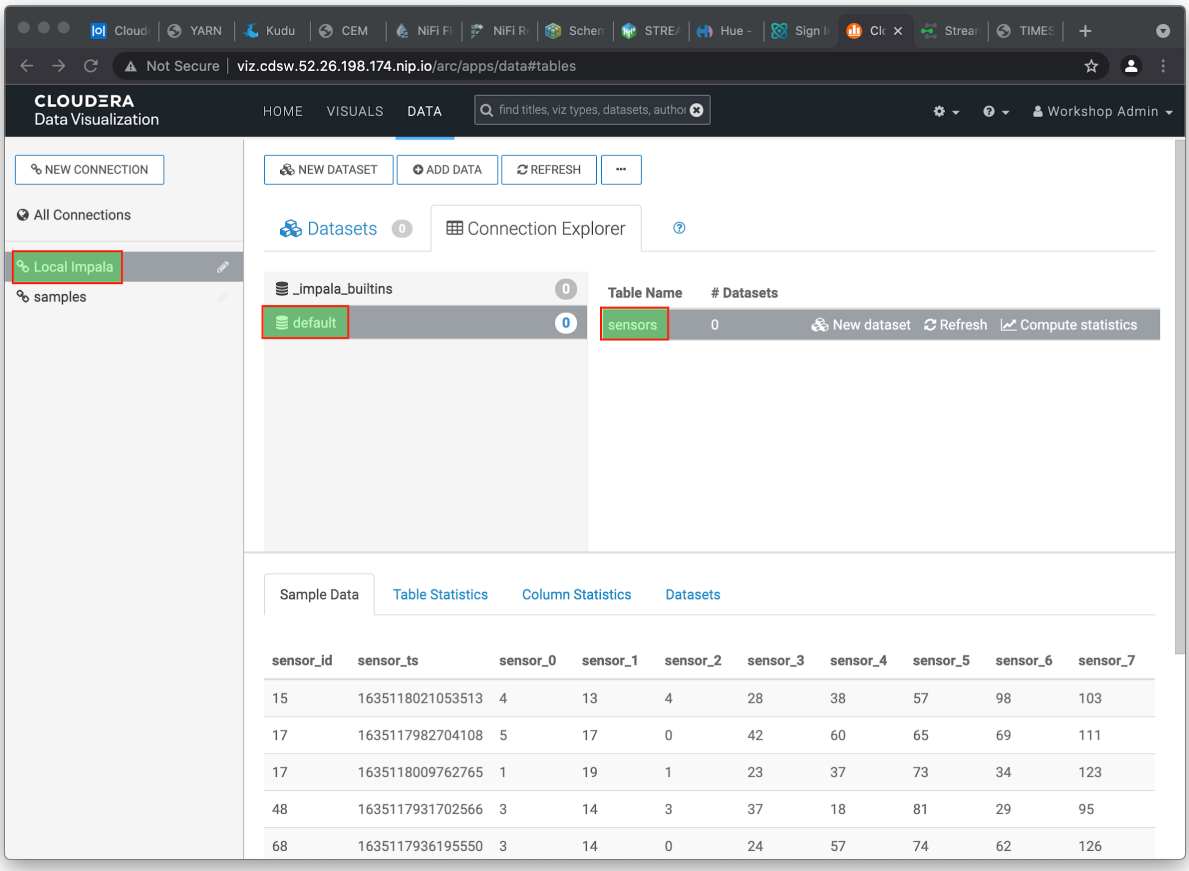
TEST CANCEL CONNECT

6. Click on CONNECT.

Lab 3 - Exploring the data

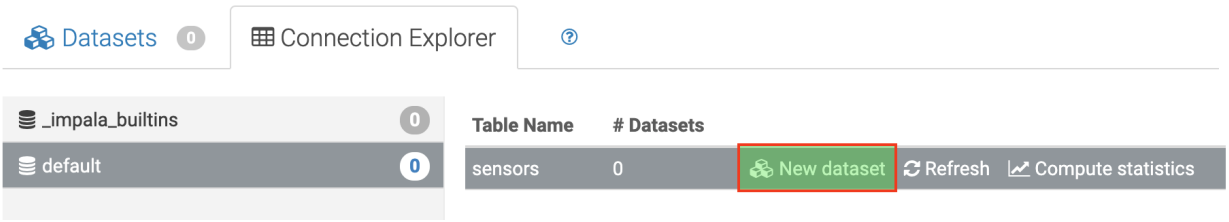
Cloudera Data Visualization provides a Data Explorer tool that enables you to explore, transform and create views of the data to suit your needs. In this lab you will look at the data available in Kudu and prepare it for your dashboard.

1. Select the newly created Local Impala connection, which you can see on the left-hand pane.
2. Select the Connection Explorer tab, then the default database and finally the sensors table. A preview with sample data will be loaded.

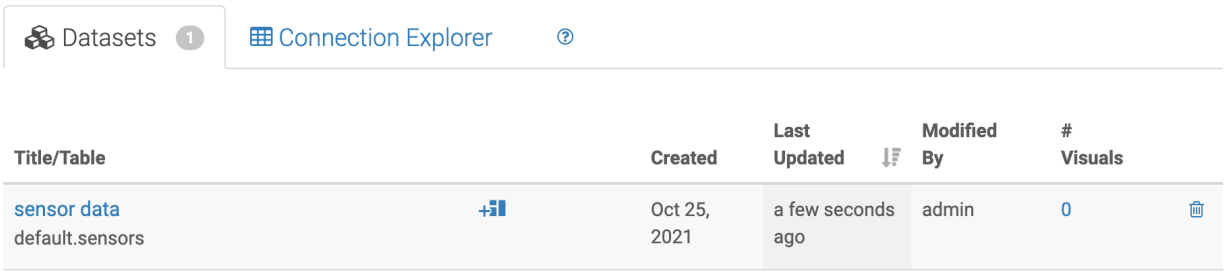


You can see in the data sample that the sensor_ts column contains the timestamp in microseconds. For your dashboard you need to convert these values into seconds instead. In the next steps you will create a new dataset and make the necessary data adjustments.

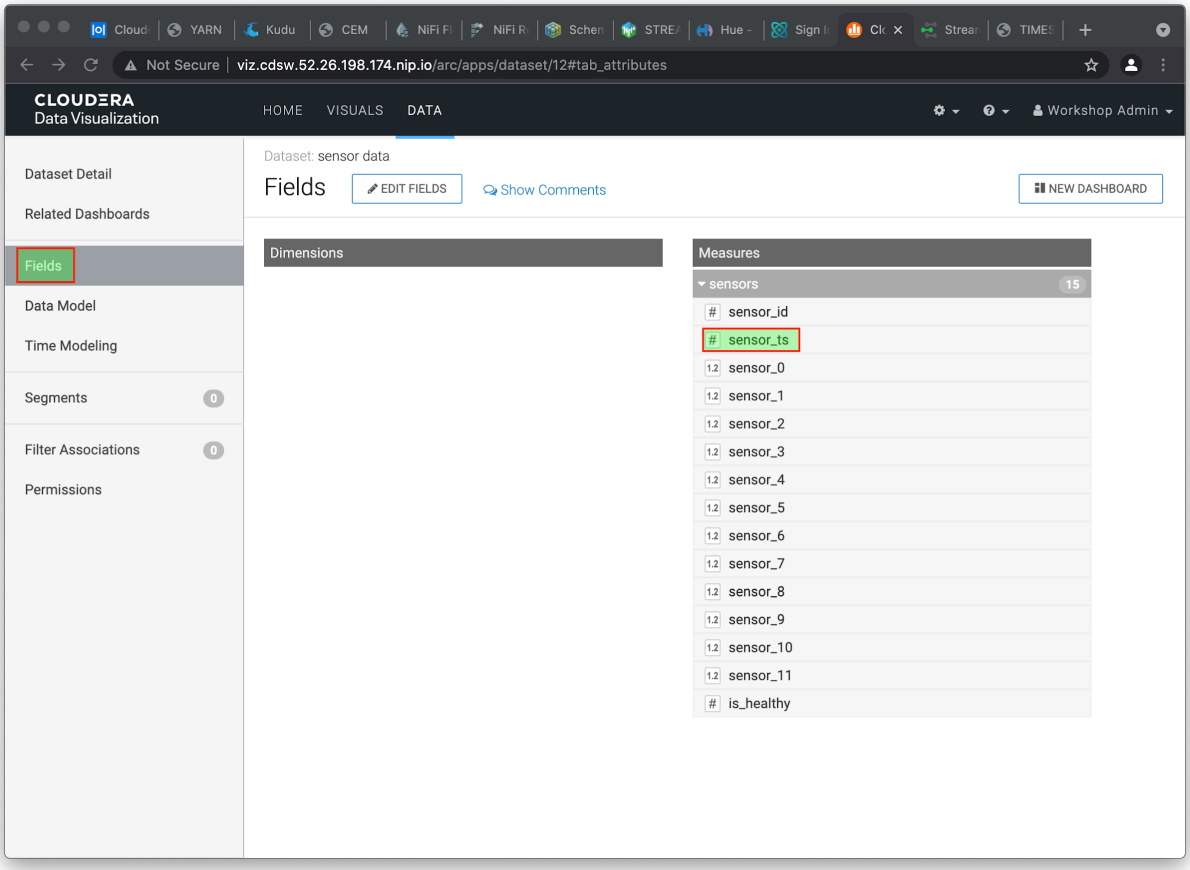
3. Click on the New dataset option besides the sensor table. Name the dataset "sensor data"



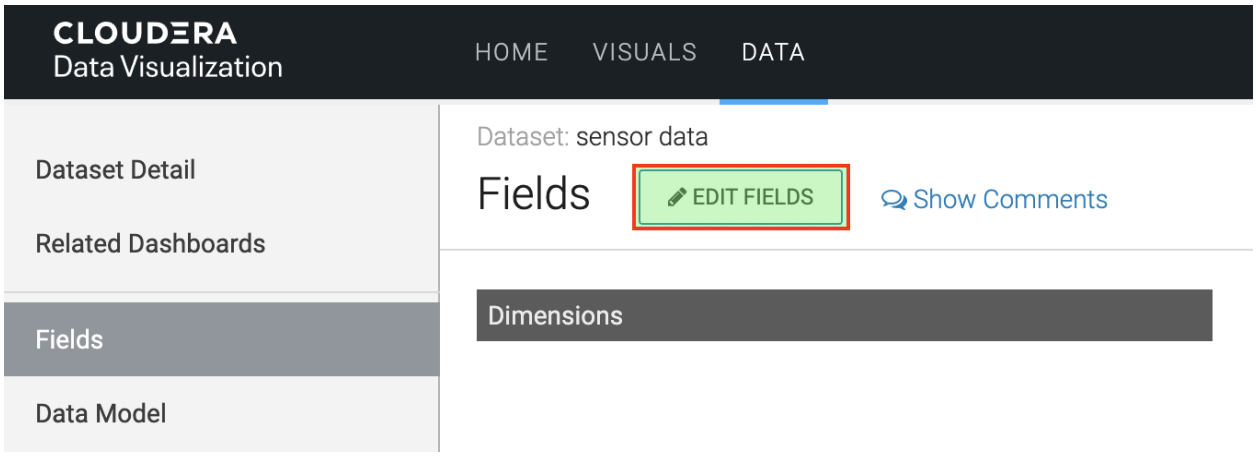
A new dataset will be created and displayed under the Datasets tab:



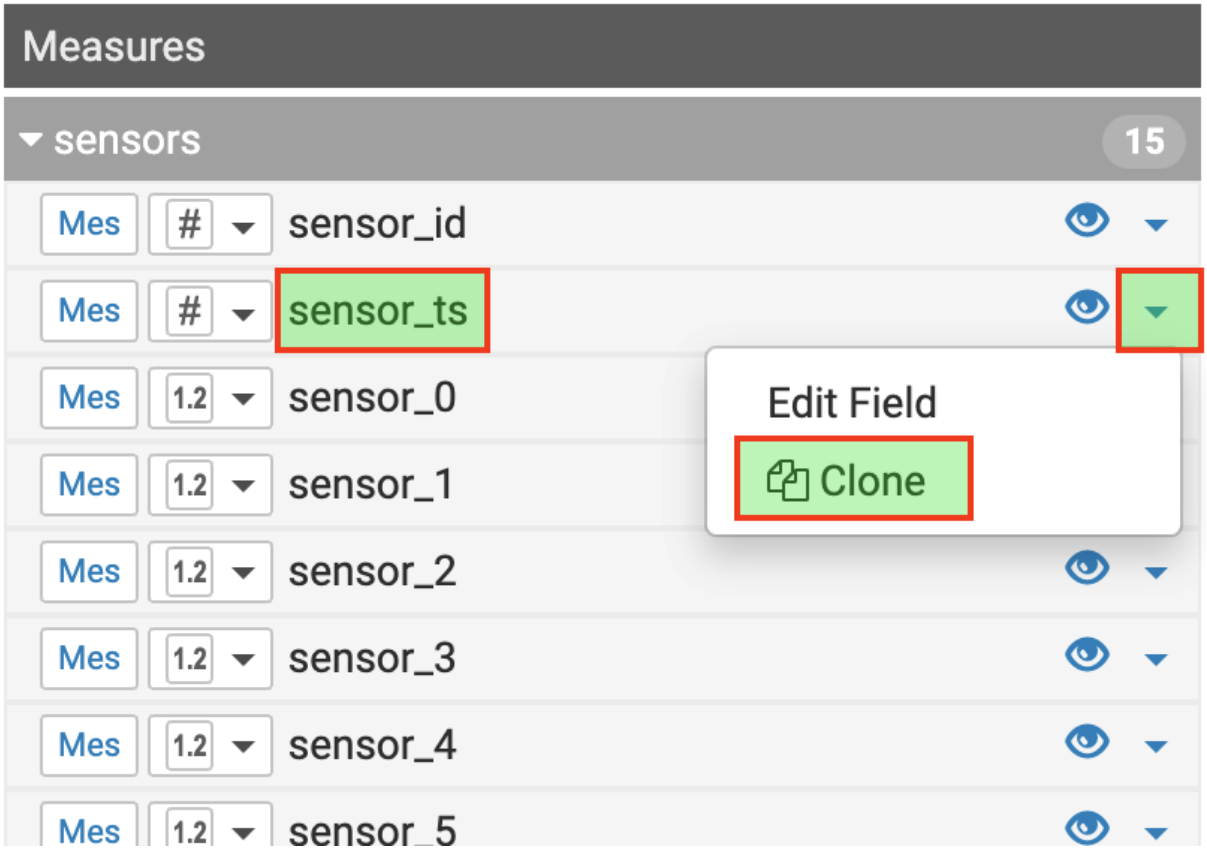
4. Click on the dataset to open it and select the Fields tab. You will notice that DataViz didn't automatically detect any dimension for the dataset.
Since the sensor_ts column is of a numeric type, and not a date/time, which is indicated by the # icon besides the field name, it was classified as a measure rather than a dimension. You will fix in the next steps.



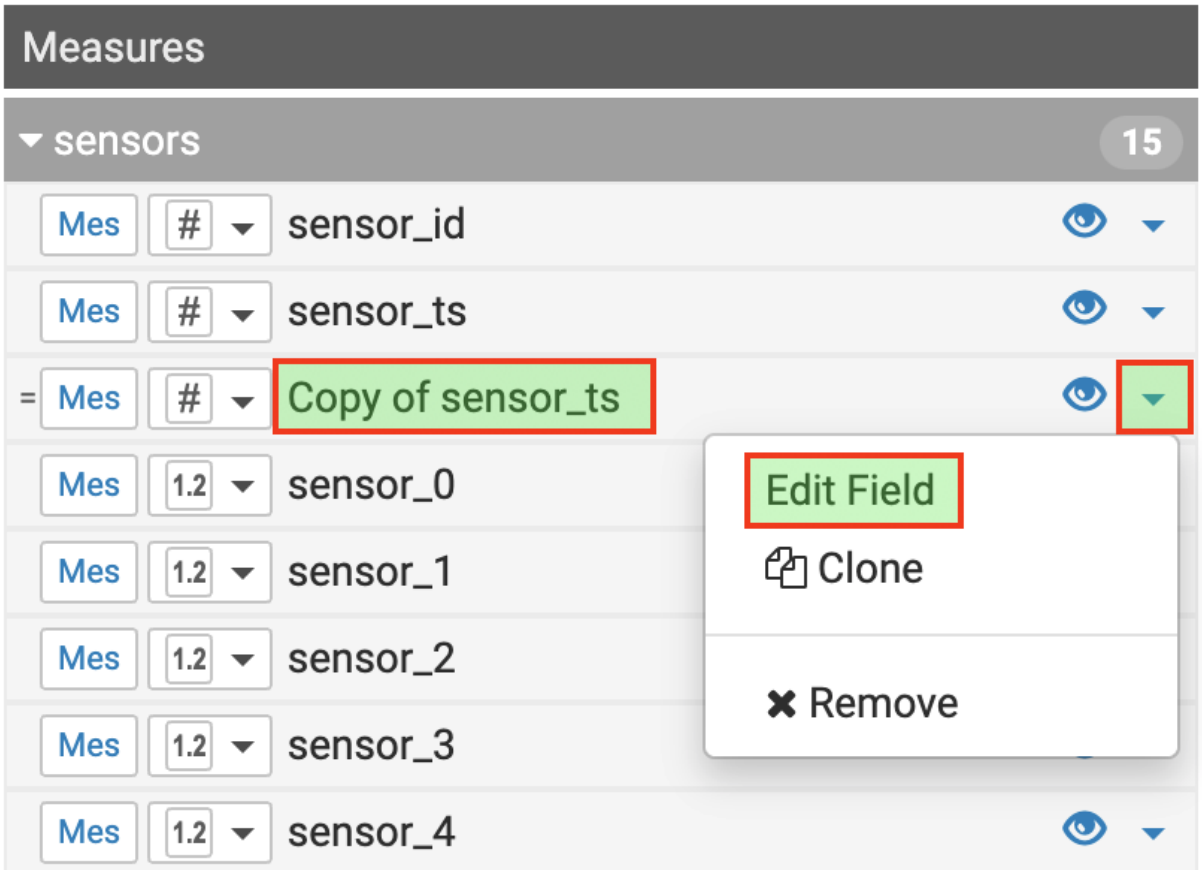
5. You need to convert the numeric fields from microseconds to seconds and convert it to a TIMESTAMP data type. In order to do this, click on the EDIT FIELDS button.



6. In the Measures list, find the sensor_ts measure, open its drop-down menu and click on Clone. A new measure Copy of sensor_ts will appear.



7. Open the drop-down menu for this new measure, and select Edit field.



8. In the Edit Field Parameters window, change the following:

In the Basic Settings tab:

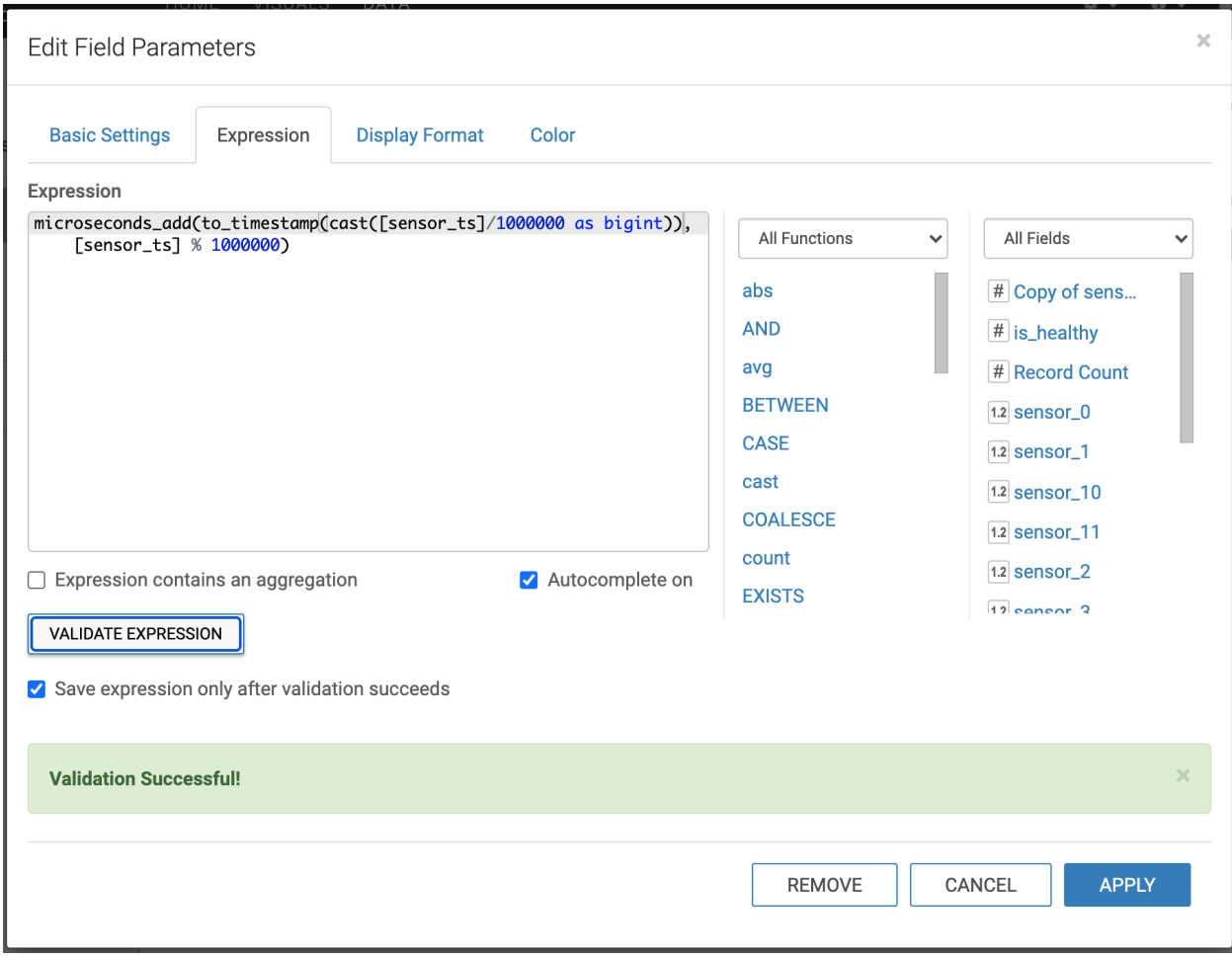
Display Name: sensor_timestamp

i. Category: Dimension

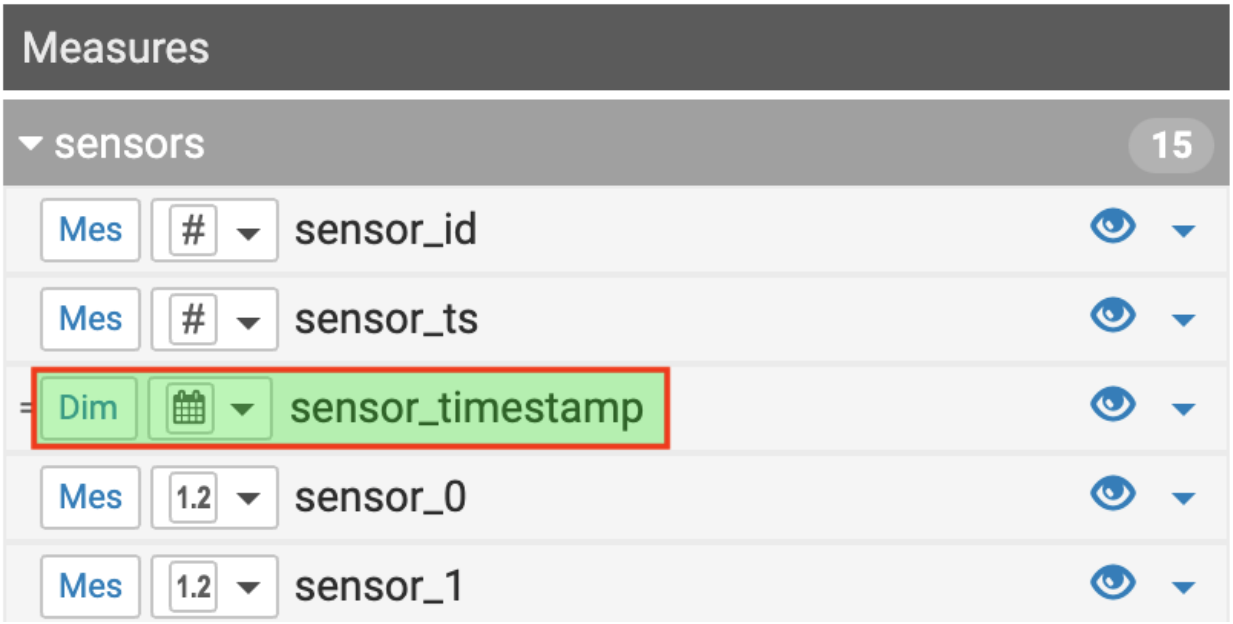
ii. In the Expression tab, enter the following expression:
microseconds_add(to_timestamp(cast([sensor_ts]/1000000 as bigint)), [sensor_ts] % 1000000)

iii. Validate the expression by clicking on VALIDATE EXPRESSION.

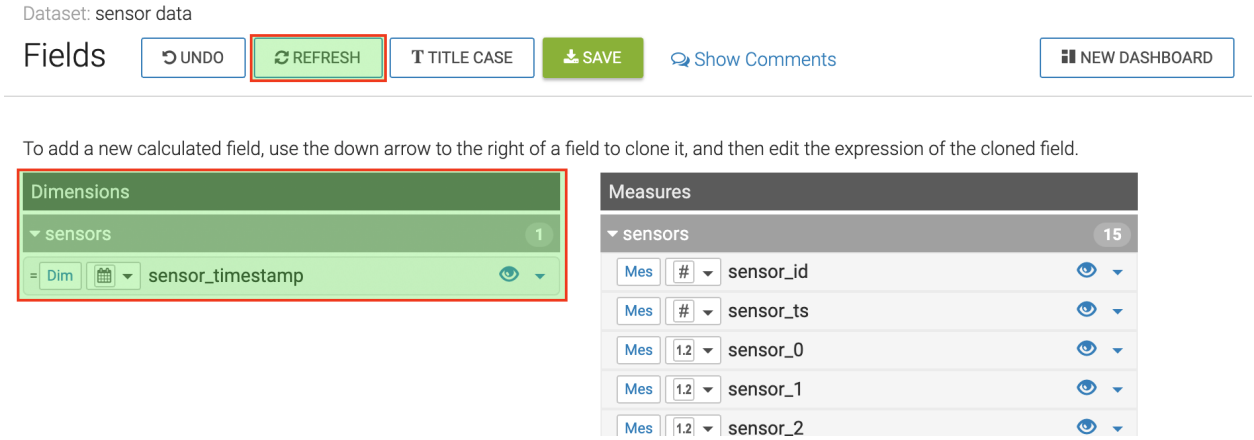
iv. Click APPLY to save the changes



9. You will notice that the category (Dim), data type (calendar icon) and field name were updated. The field still shows up in the Measure category, though.



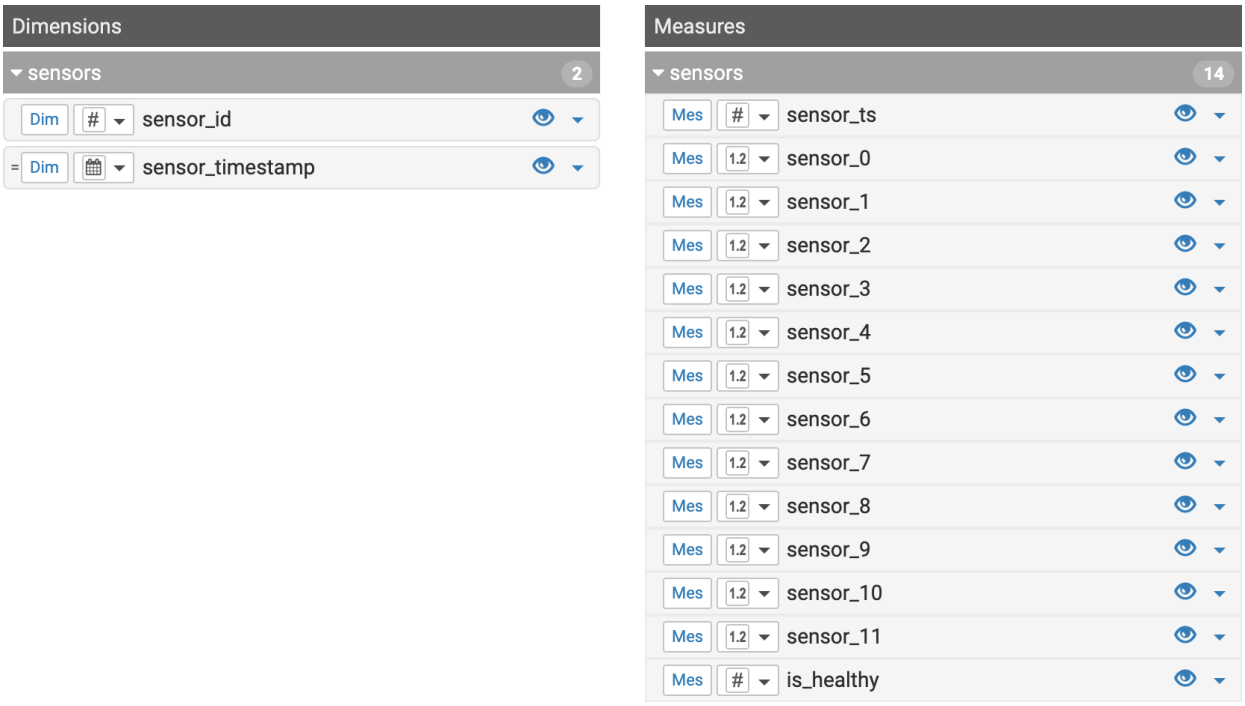
This is just refresh issue. Click on the REFRESH button at the top and you should see the sensor_timestamp field "jump" to the Dimensions category.



10. The sensor_id field is also a dimension and needs to be moved to the correct category. To do this, find the sensor_id field under the Measures category and click on the 

icon to toggle it to 

. Click on the REFRESH button again and you should see the following structure for your dataset:



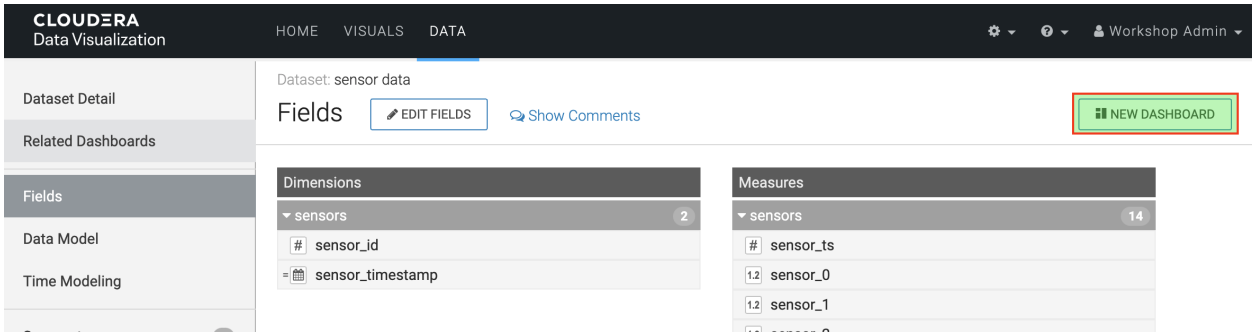
11. Save you changes by clicking the green Save button.

You have just created a dataset to feed your dashboard and performed the necesssary adjustments for your data source. In the next lab you will create the dashboard from it.

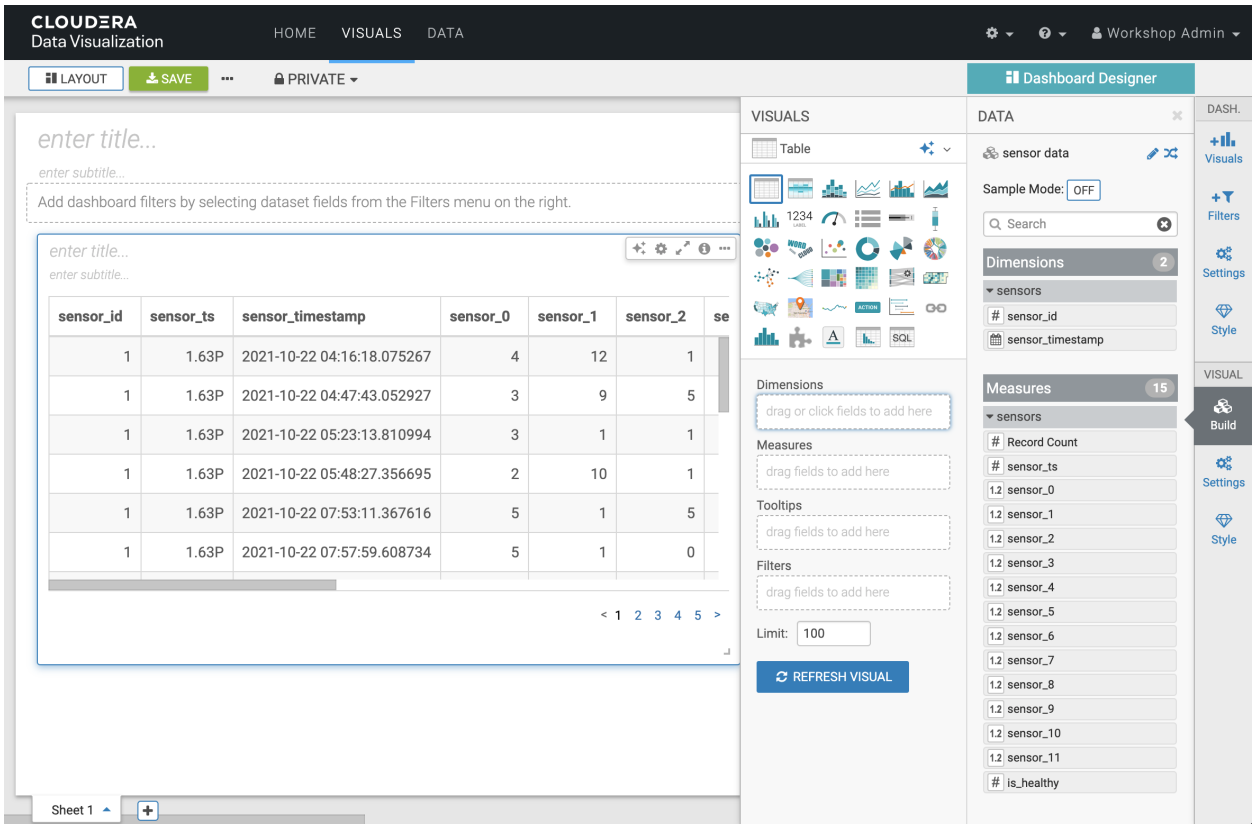
Lab 4 - Creating a dashboard

You have everything ready now to start building your dashboard. Let's jump straight into it:

1. On your dataset page, click on the NEW DASHBOARD button.

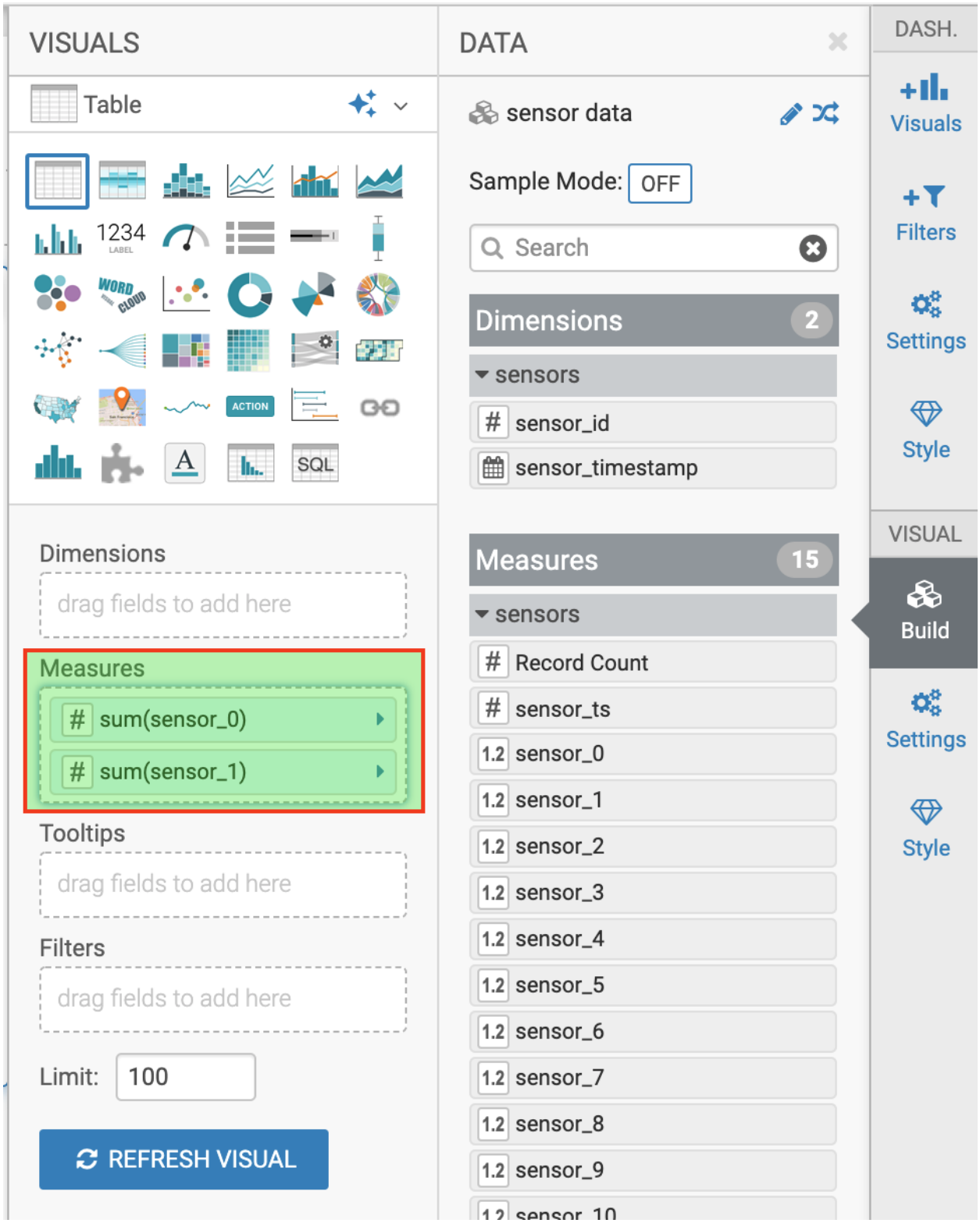


2. Since we initiated the dashboard creation from the dataset page, will you notice that the dashboard is already created by default with a "table visual" displaying all fields of the dataset.



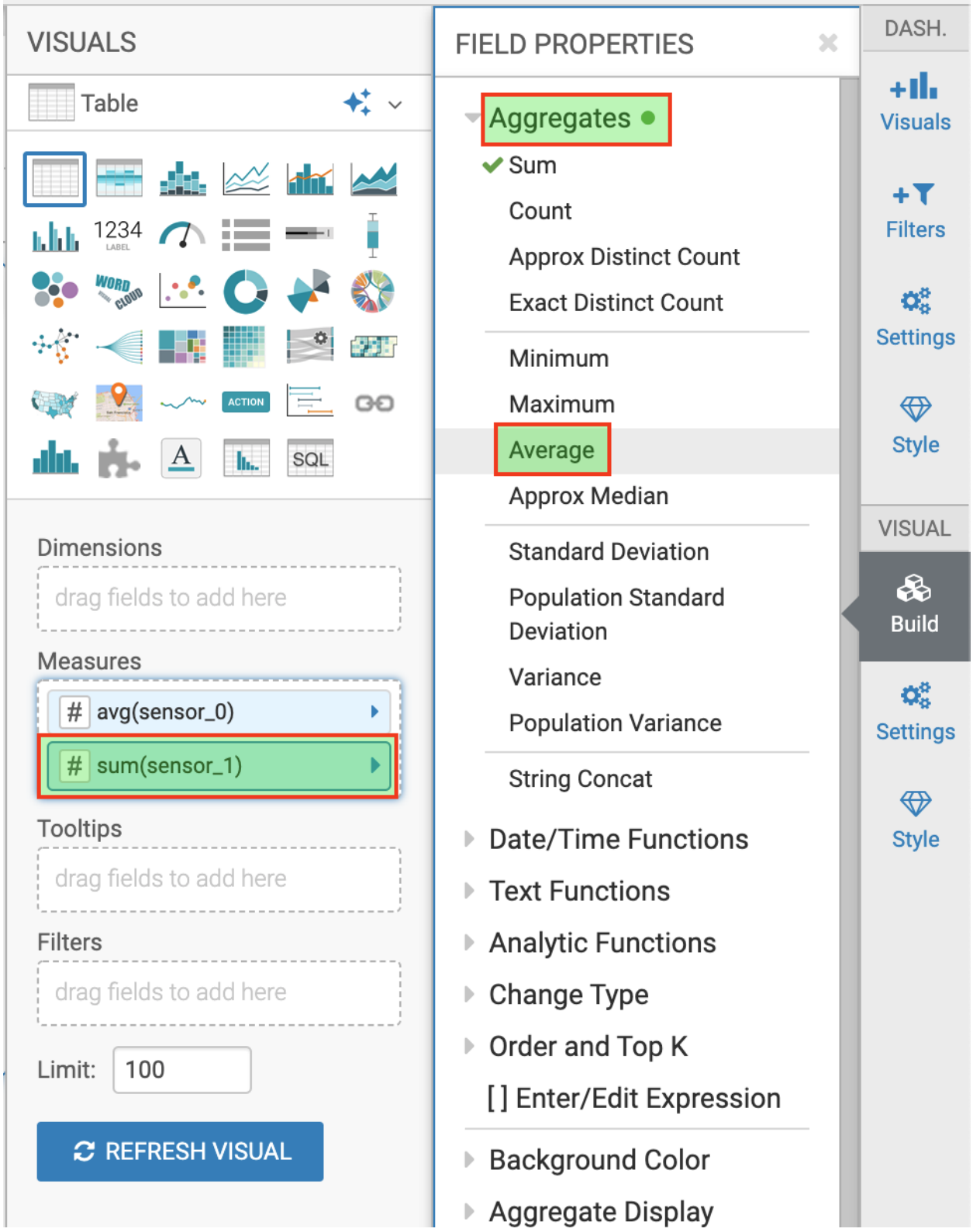
3. Click on the table visual to ensure it is selected (you see a blue border around the visual when it is selected). With the table visual selected, click on the Build tab on the right.

4. Click on the Measures input box to select it. Then click on the fields sensor_0 and sensor_1 from the Measures list. These fields will be added to the Measures input box.



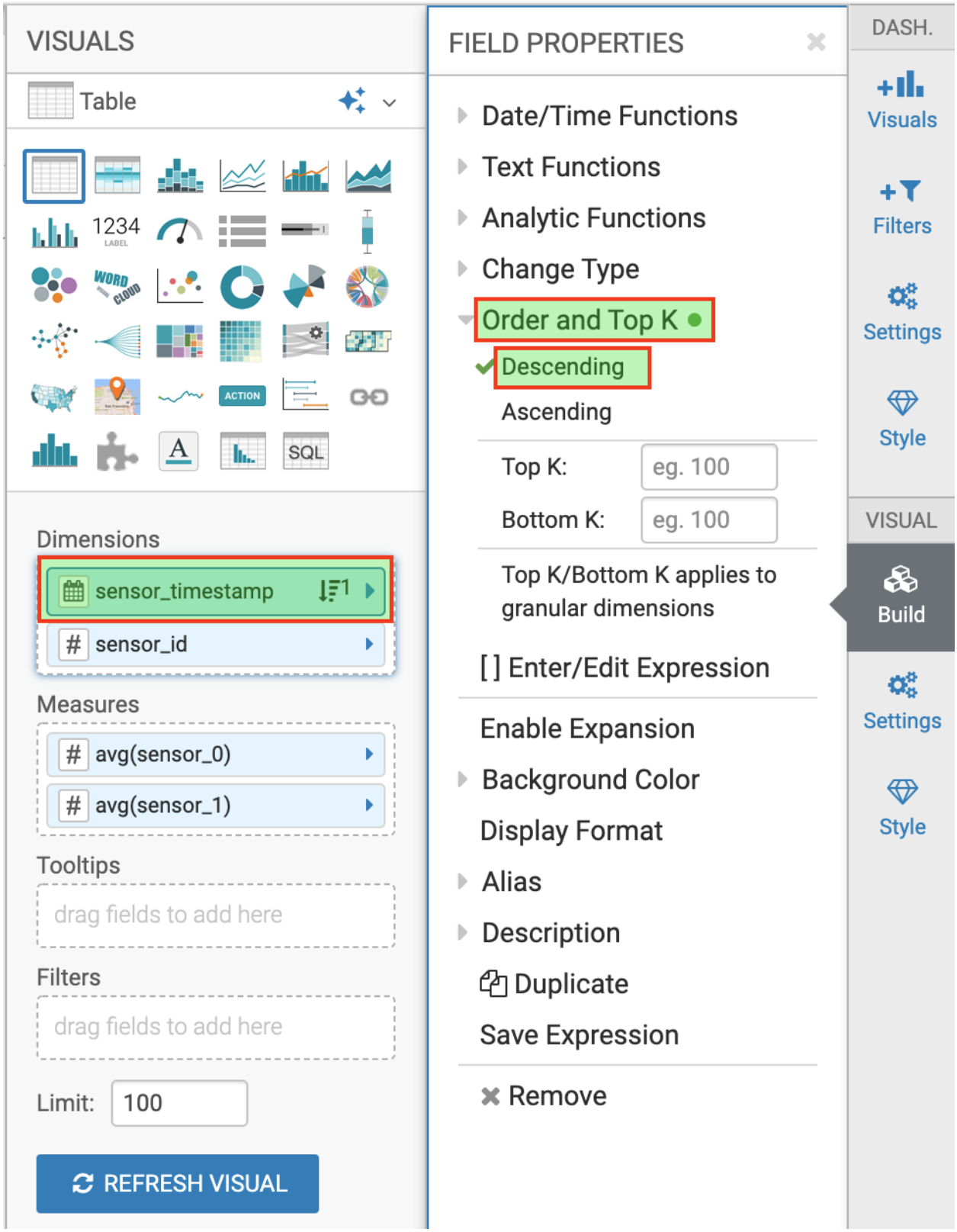
5. The measures are added, by default, with the sum() aggregation. Change it to avg() by selecting each one of the newly added measures and selecting Aggregates > Average.

Ensure this is done for both measures.



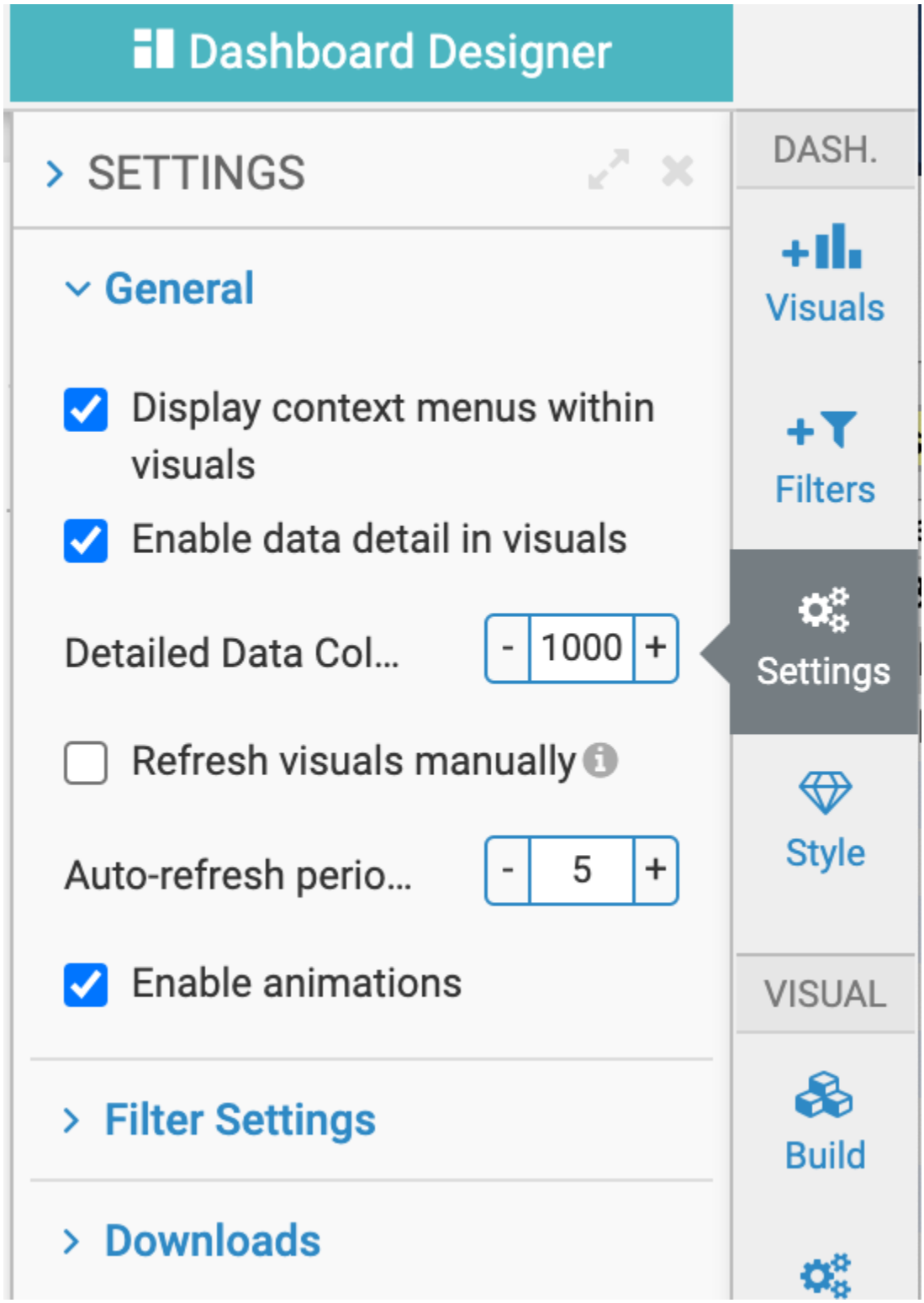
6. Click on the Dimensions input box to select it. Then click on the fields sensor_timestamp and sensor_id from the Dimensions list. These fields will be added to the Dimensions input box.
7. Highlight sensor_timestamp field in the Measures input box and select Order and Top K > Descending. This will show the values in the table visual in descending order with the

newest sensor readings on top.



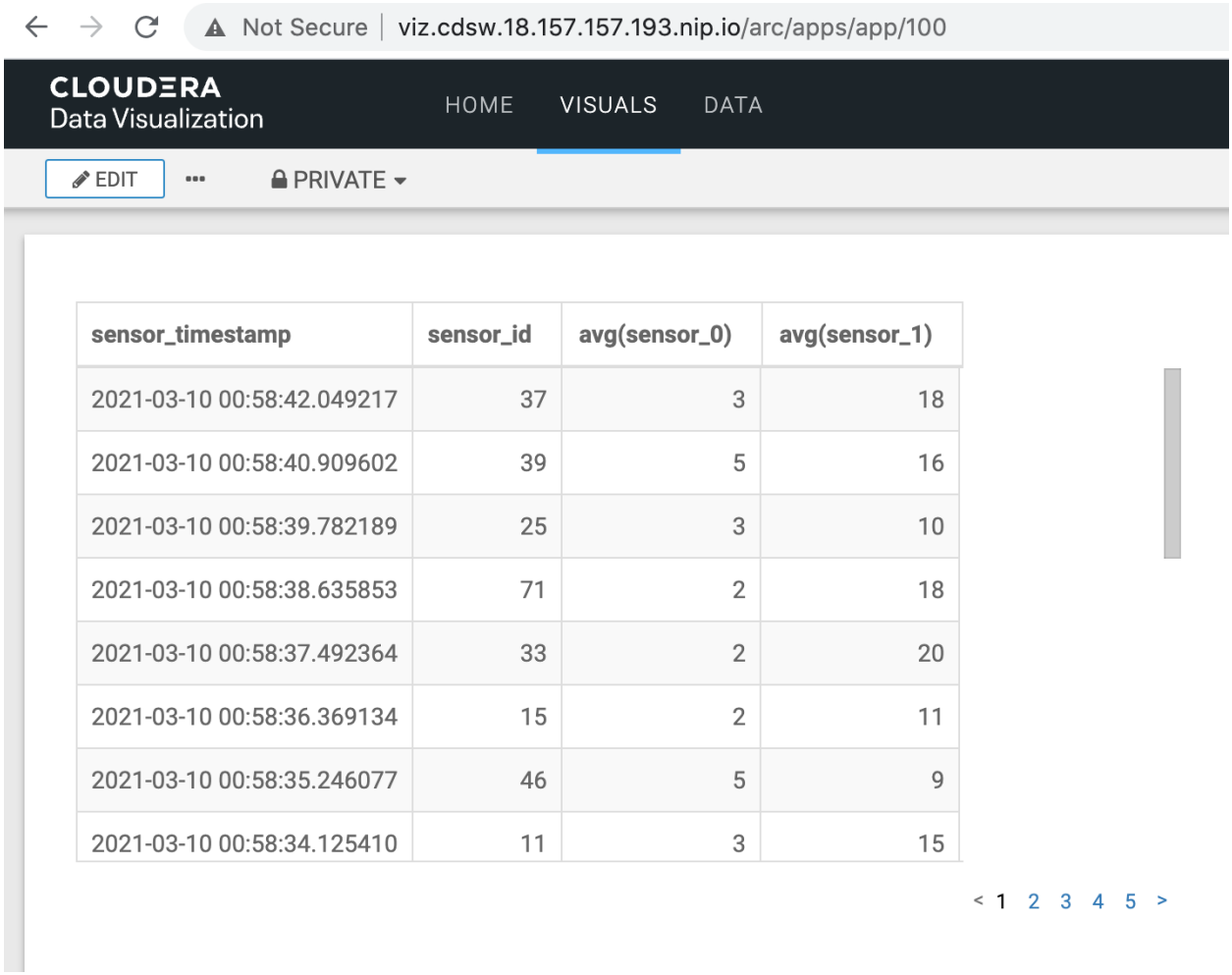
8. Click on Refresh visual to update the visual with the latest changes.

9. Finally, select the Settings tab on the right of the screen and change the value for Auto-refresh period (sec) to 5.



10. Click on the Save button at the top of the dashboard to save the changes and click View to enter view/publish mode. This is what your dashboard consumers will see: the sensor

reading coming in through the streaming pipeline, displayed in a real-time dashboard, updating automatically.

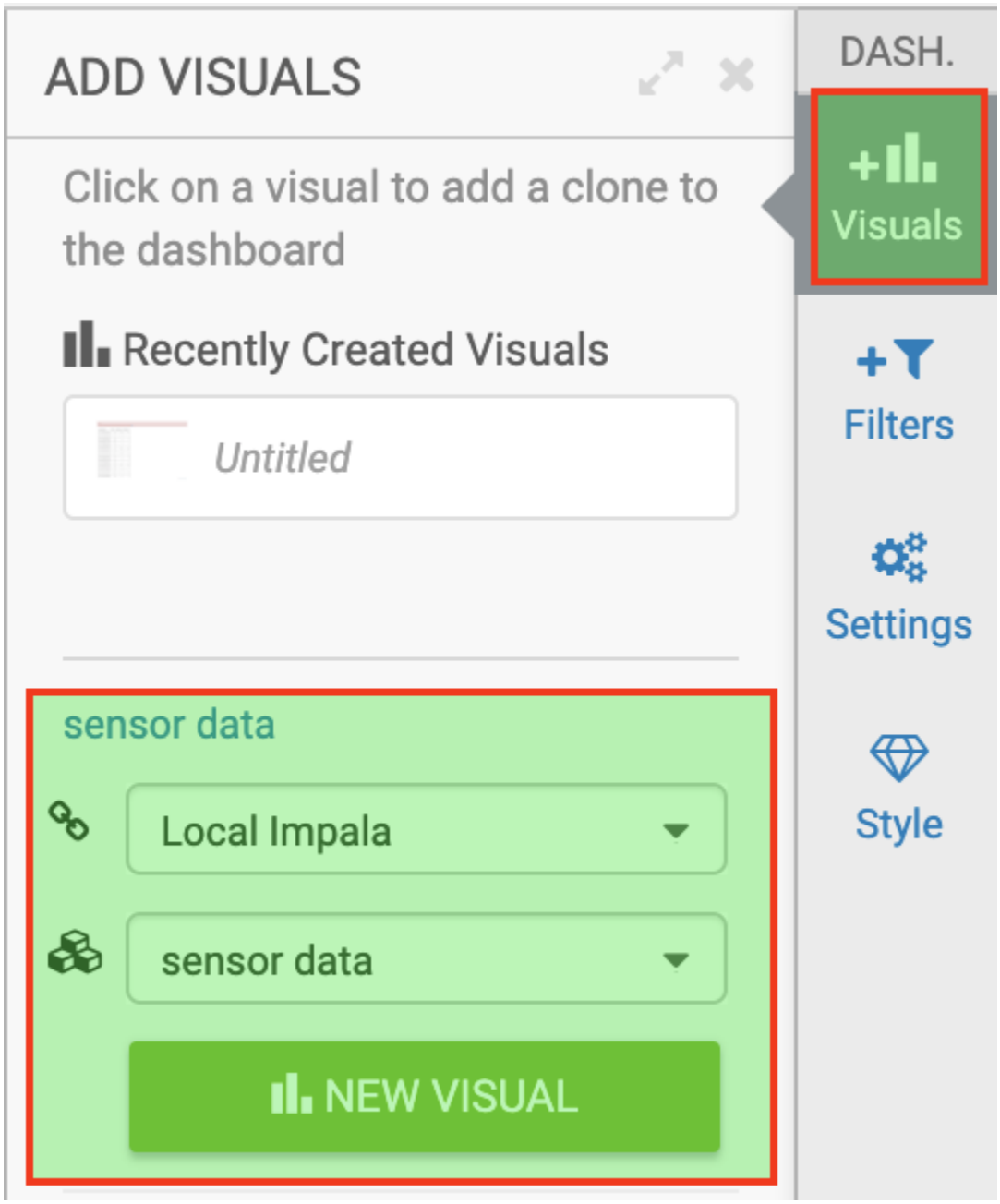


Lab 5 - Adding a chart

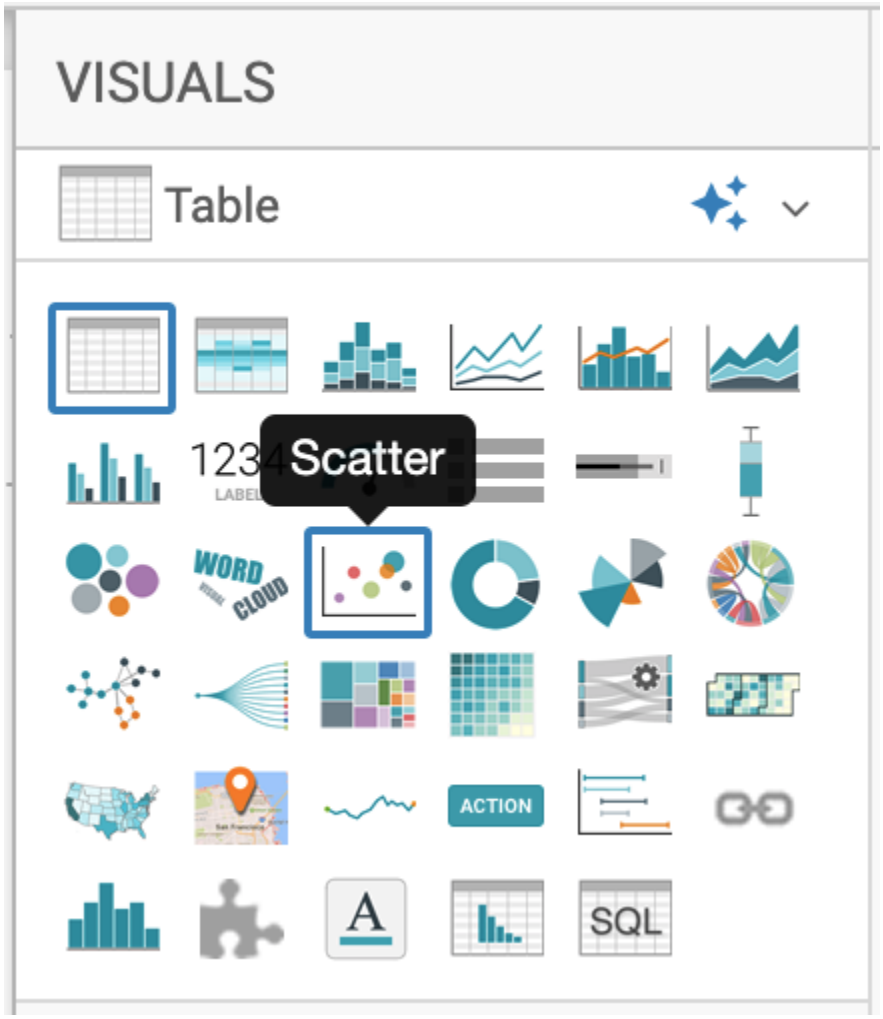
Dashboards are usually synonym with graphs and charts. Cloudera Data Visualization comes with a myriad of charts types to help visualize your data. In this lab you'll add a simple bar chart to your dashboard to make it more interesting.

1. On the view mode dashboard above, click on the EDIT button to go back into editing mode.

2. Click on the Visuals tab on the right. Ensure the Local Impala connection and the sensor data dataset are selected and click on the NEW VISUAL button.



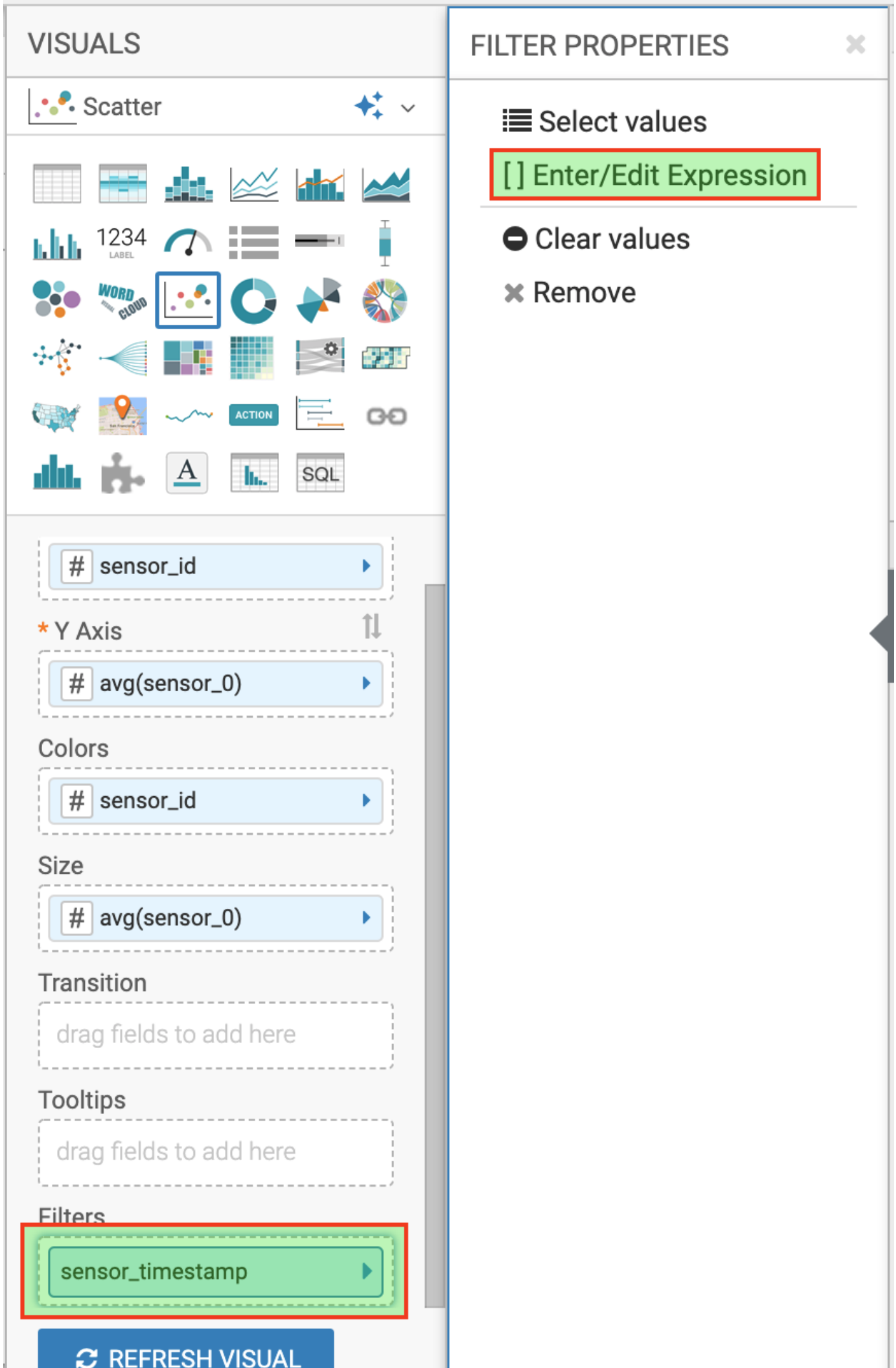
3. On the Visuals tab, select the Scatter visual type:



Based on what you learned in the previous lab, enter the following properties:
X Axis: sensor_id
Y Axis: avg(sensor_0)
Colors: sensor_id
Size: avg(sensor_0)

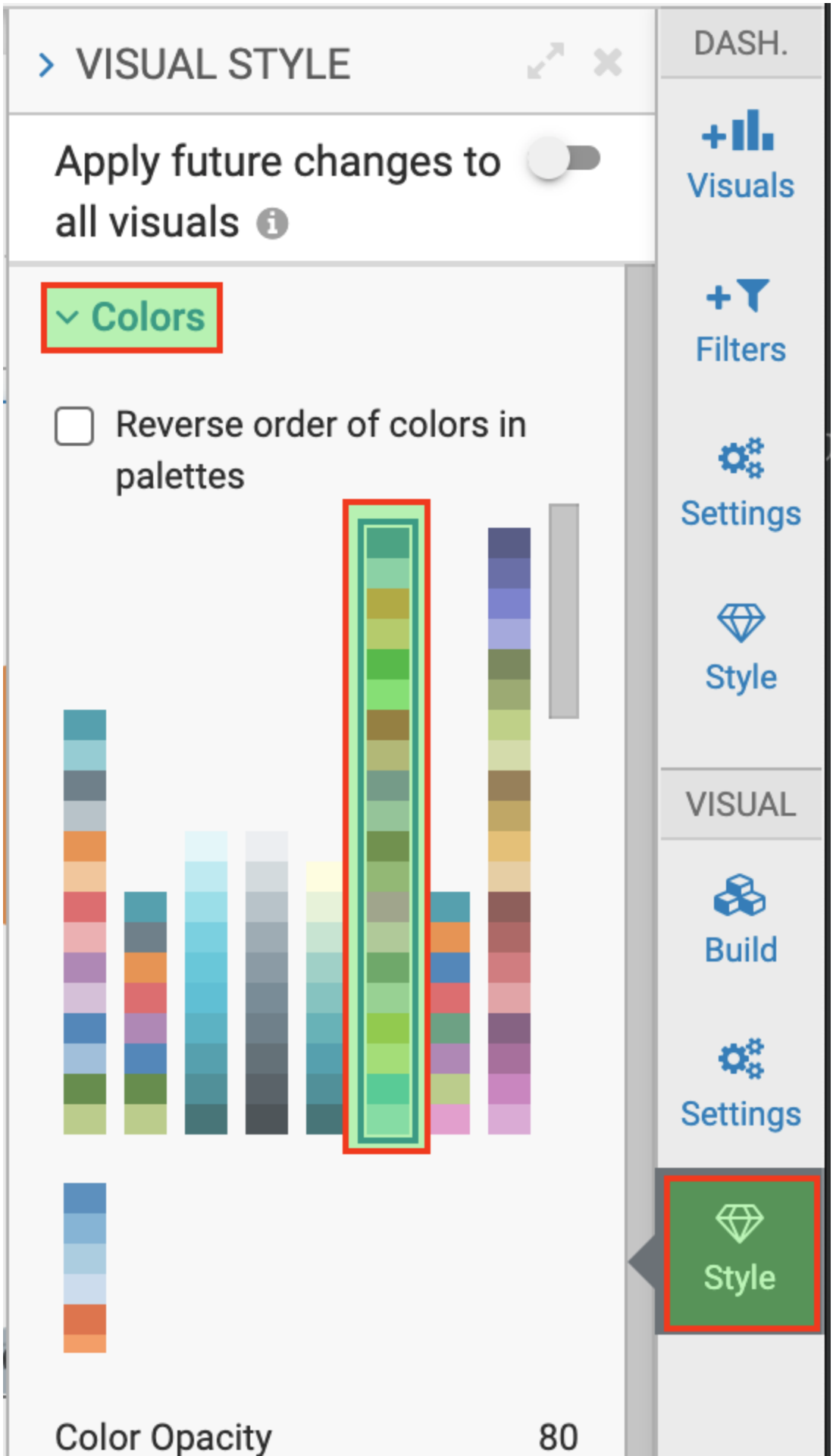
4. Filters: sensor_timestamp

5. Click on the sensor_timestamp filter to select it and then click on [] Enter/Edit Expression.

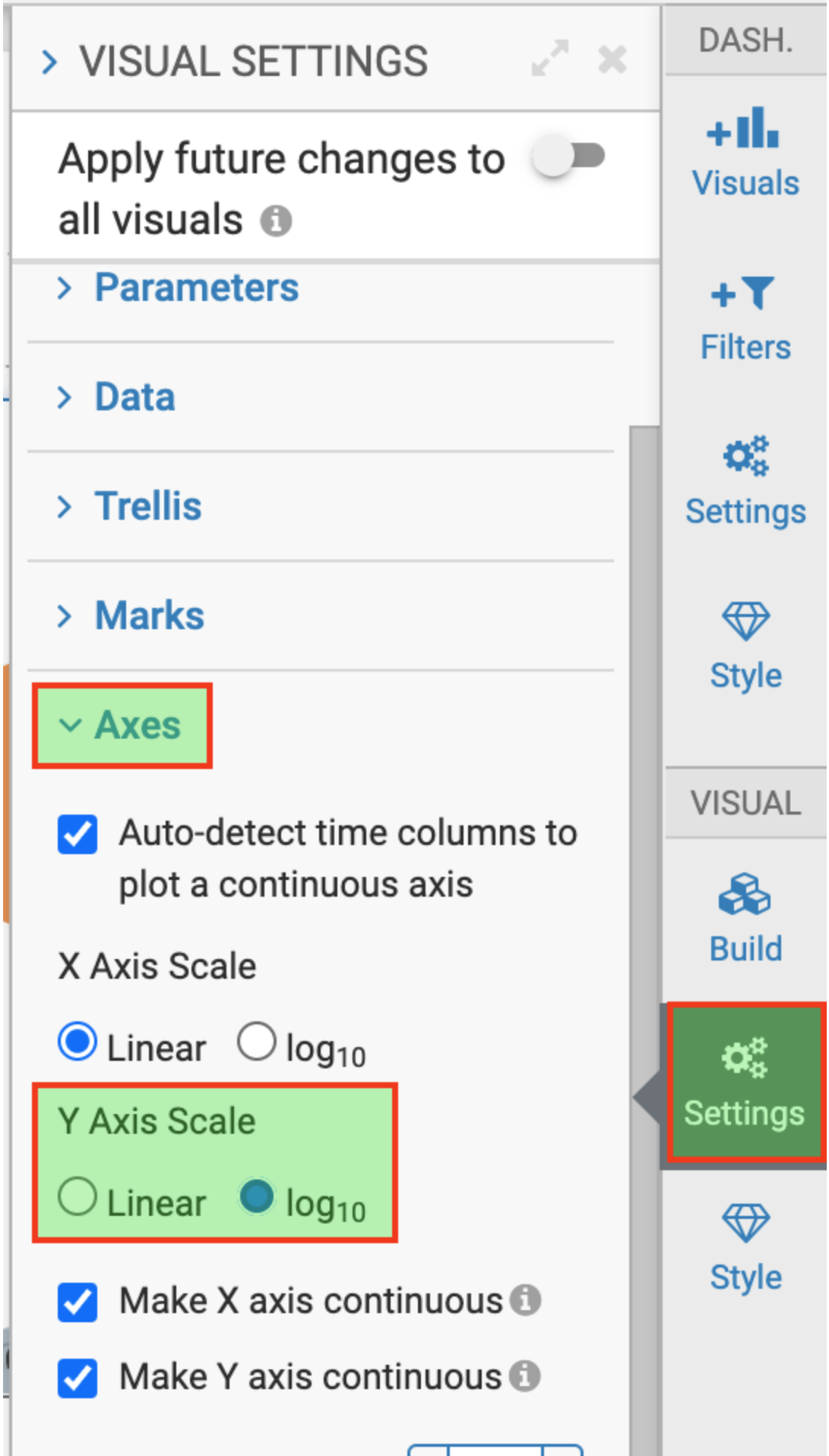

VISUALS
Scatter
1234
LABEL
WORD CLOUD
ACTION
SQL
sensor_id
* Y Axis
# avg(sensor_0)
Colors
sensor_id
Size
avg(sensor_0)
Transition
drag fields to add here
Tooltips
drag fields to add here
Filters
sensor_timestamp
REFRESH VISUAL
FILTER PROPERTIES
Select values
[ ] Enter/Edit Expression
Clear values
Remove

6. Enter the following expression in the Enter/Edit Expression window to limit the data shown in the chart to the last minute of data received. This will create a chart over a rolling window of 1 minute.
   [sensor_timestamp] > seconds_sub(now(), 60)
7. Validate the expression and click Save.

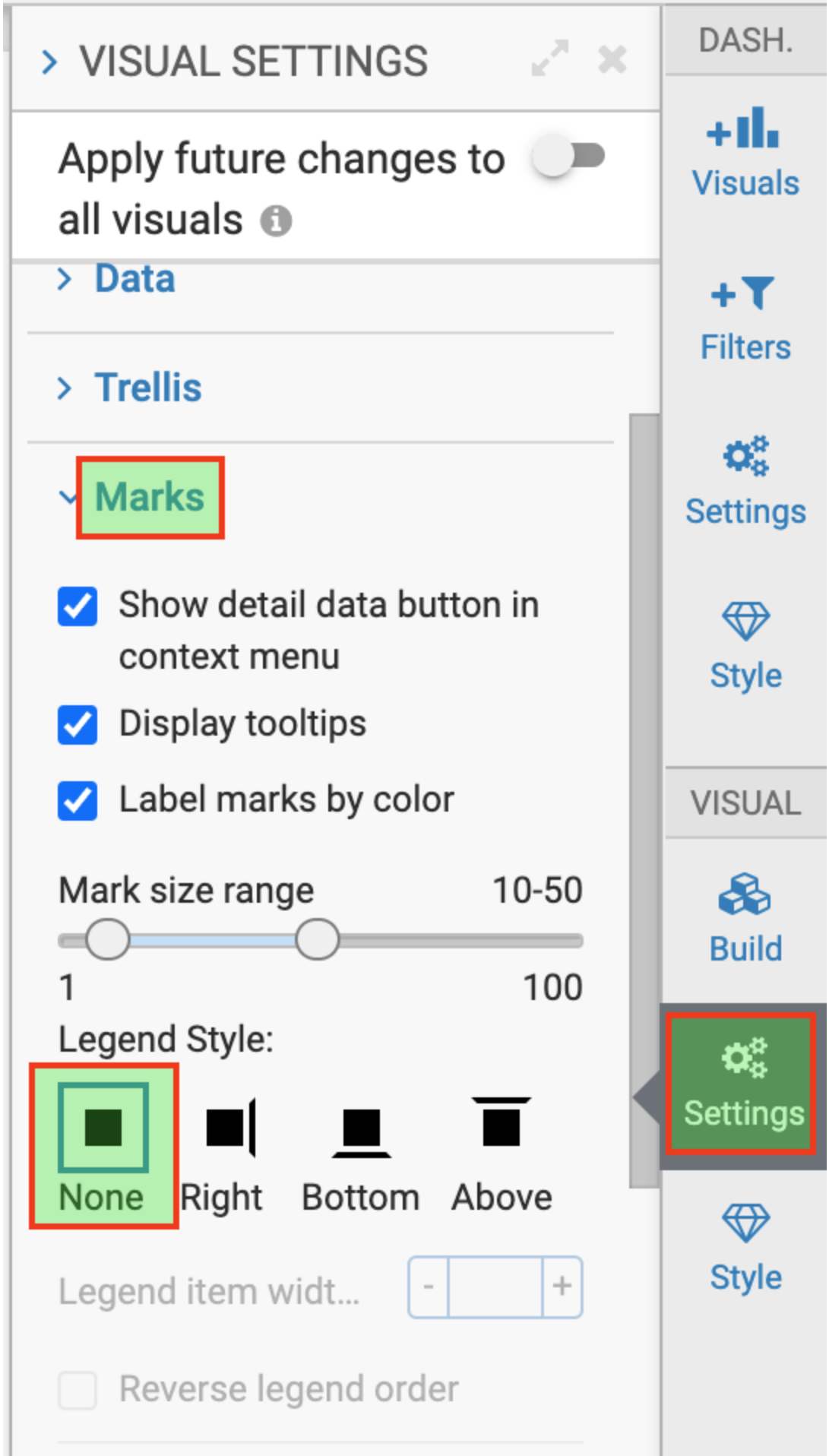
8. Click on VISUAL > Style on the right-hand tab, and select a colorful palette in the Colors section.


VISUAL STYLE
Apply future changes to all visuals
Colors
Reverse order of colors in palettes
Color Opacity
80
DASH.
Visuals
Filters
Settings
Style
VISUAL
Build
Settings
Style

9. Click on VISUAL > Settings on the right-hand tab, and set the Y Axis Scale to log10 in the Axes section.

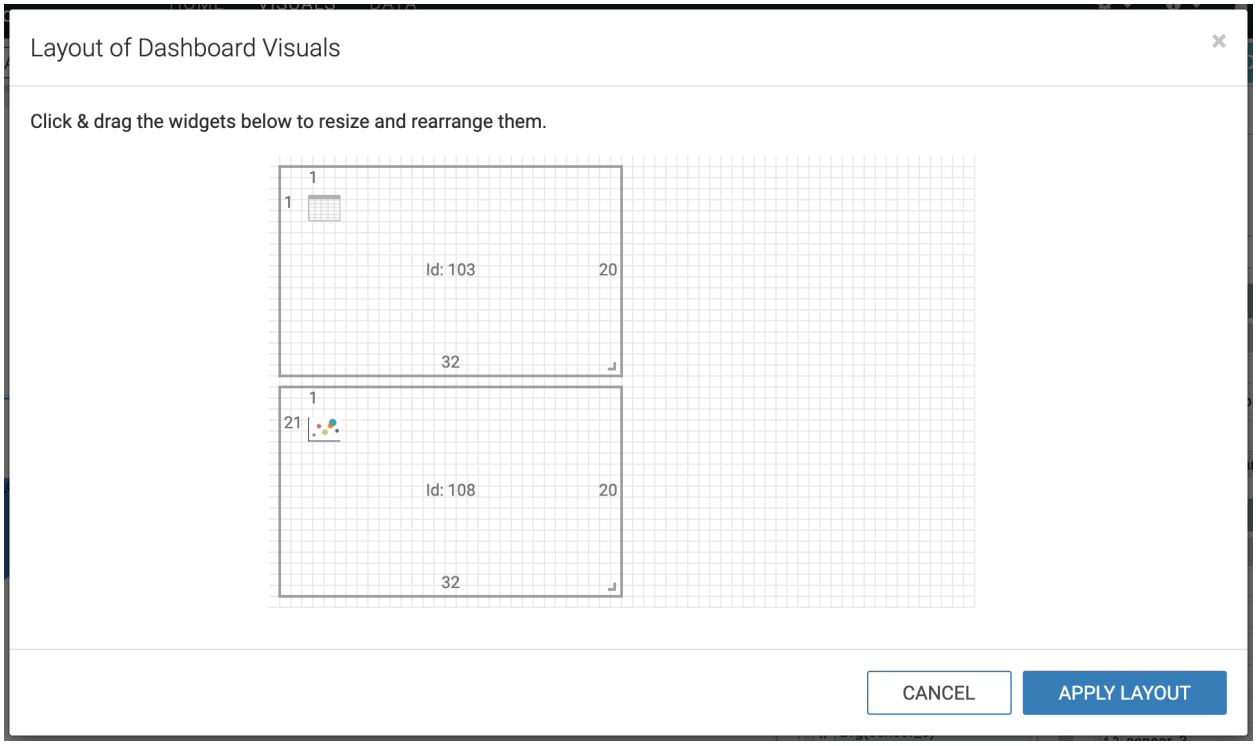

VISUAL SETTINGS
Apply future changes to all visuals
Parameters
Data
Trellis
Marks
Axes
Auto-detect time columns to plot a continuous axis
X Axis Scale
Linear
log10
Y Axis Scale
Linear
log10
Make X axis continuous
Make Y axis continuous
DASH.
Visuals
Filters
Settings
Style
VISUAL
Build
Settings
Style

10. Expand the Marks section and set the Legend style to None.

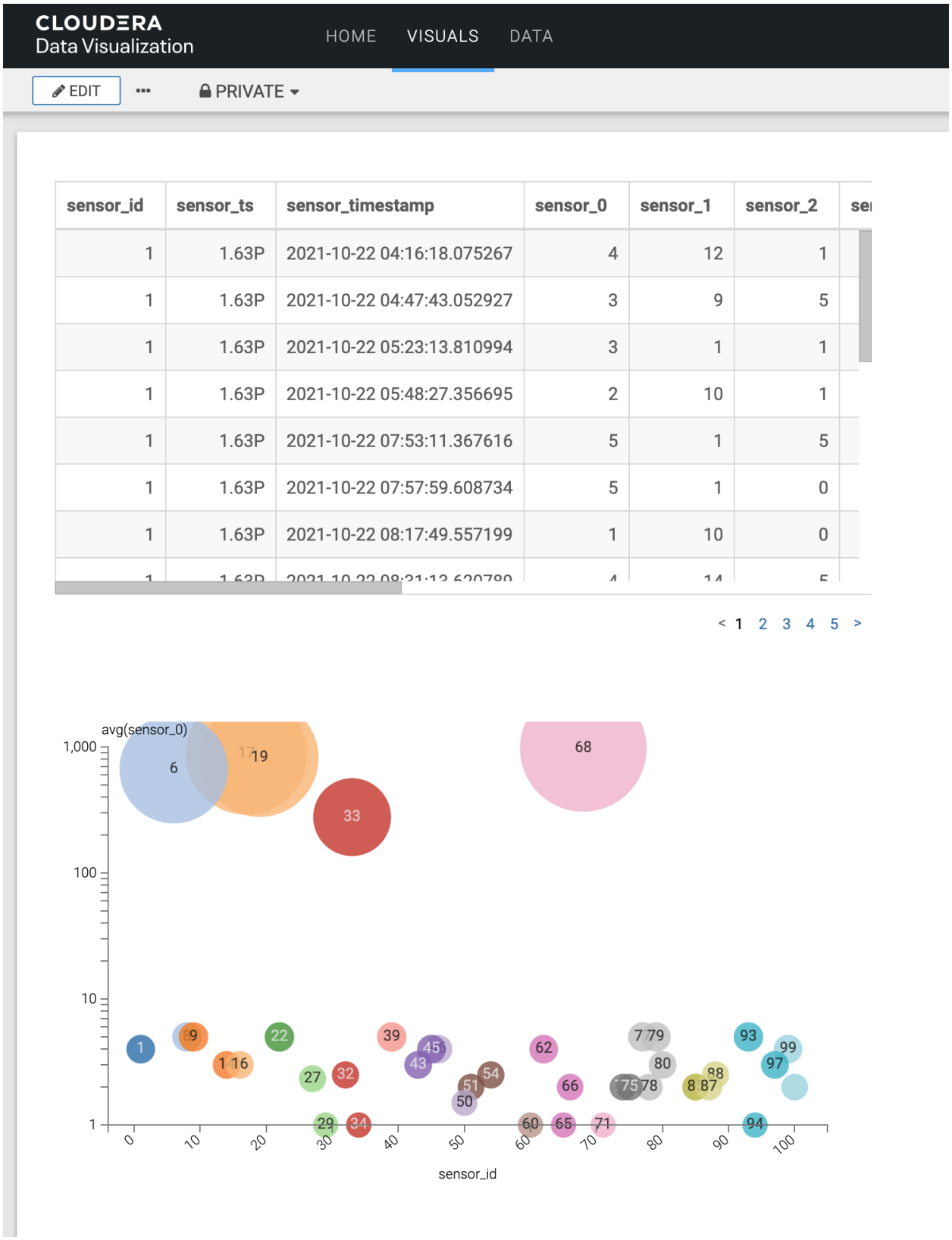

VISUAL SETTINGS
Apply future changes to all visuals
Data
Trellis
Marks
Show detail data button in context menu
Display tooltips
Label marks by color
Mark size range
10-50
1
100
Legend Style:
None
Right
Bottom
Above
Legend item widt...
Reverse legend order
DASH.
Visuals
Filters
Settings
Style
VISUAL
Build
Settings
Style

11. Click on the 

button, at the top of the Dashboard Designer to arrange the visuals in your dashboard. Drag the two visuals in the diagram to position them as you would like. Once you are done, click on APPLY LAYOUT.

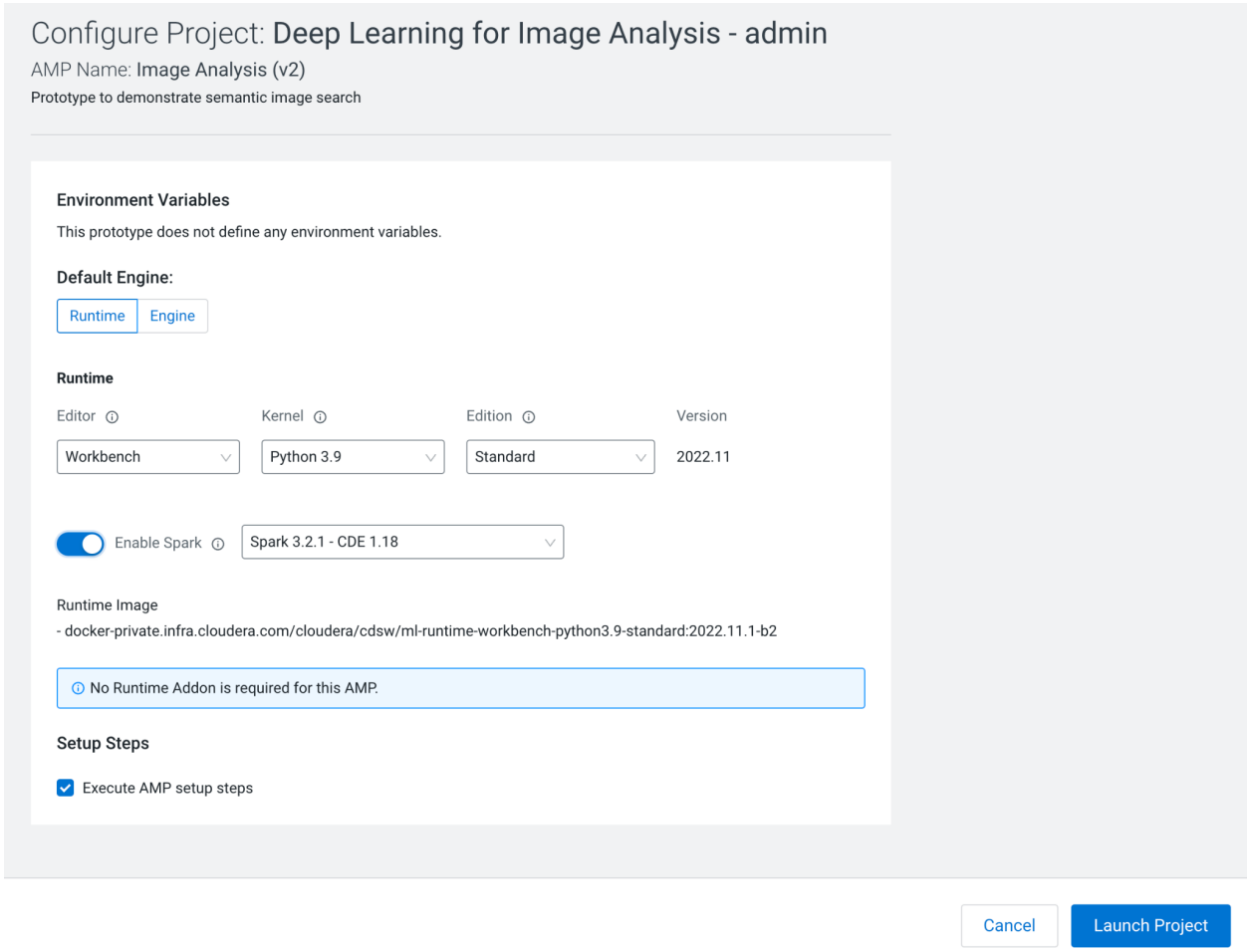


12. Click on the Save buttons to save the changes to your dashboard and then click on View to switch to the view mode and check your real-time dashboard in action:



Deep Learning Image Analysis AMP

Go to **AMPs**, choose **Deep learning for Image Analysis**, select Configure Project and click Launch Project.



Once all the AMP Setup steps are completed, click open link shown in step 4.

AMP Setup Steps

AMP Name: **Image Analysis (v2)**

Prototype to demonstrate semantic image search

Completed all steps

Step 3 Running FAISS index creation job. View details completed 1/25/2023 11:15 AM

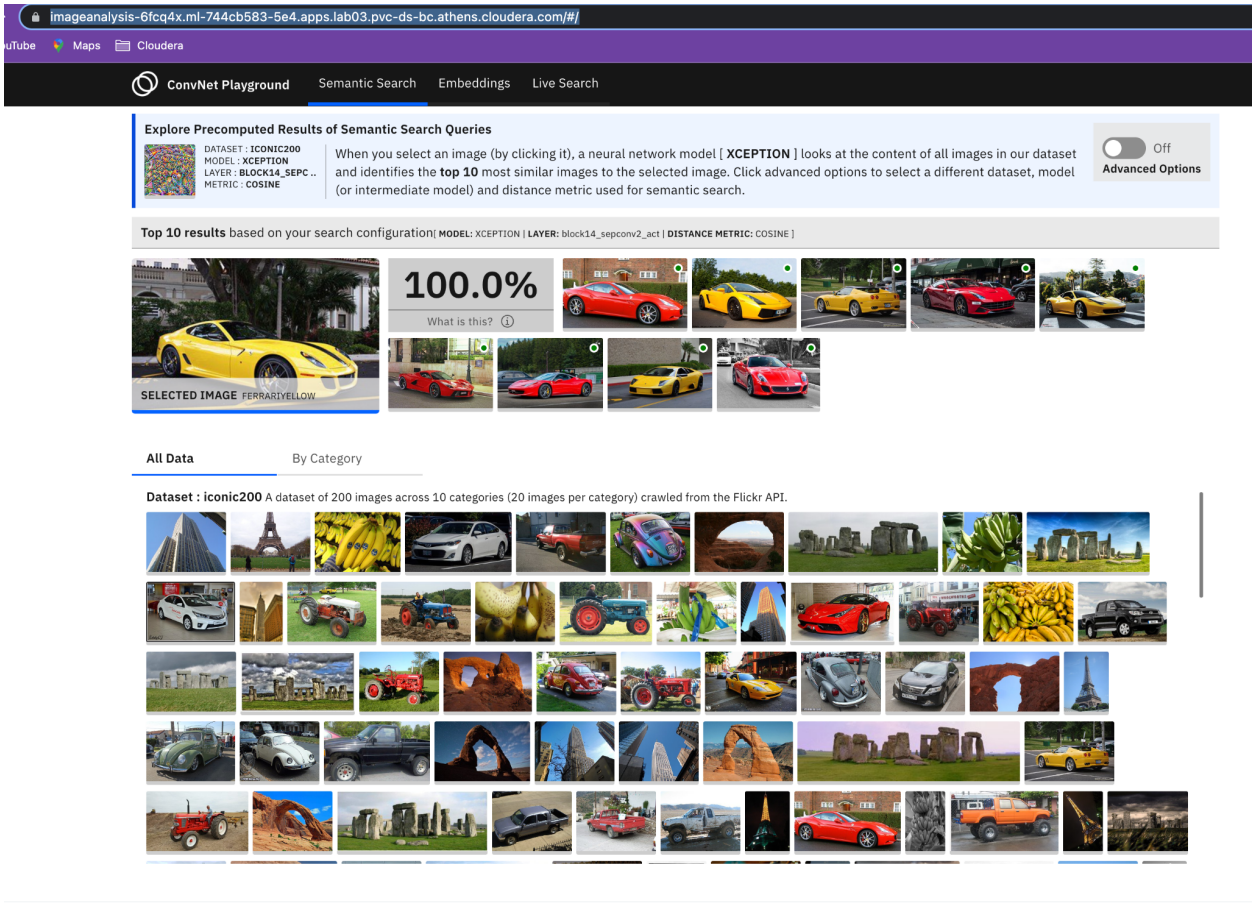
Running FAISS index creation job.

```
        index (lib.FaissIndex): index to be updated
        image_dir (str): directory of images to be added to index
        index_save_dir (str): directory to save index on disc.
    """
    features, ids = extractor.extract_from_dir(image_dir, efficientnet_model)
    print(features.shape, len(ids))
    index.add(features, id_strings=ids)
    index.save(index_save_dir)
index = create_index(fashion_images_dir, os.getcwd() + "/faiss")
update_index(index, iconic_images_dir, os.getcwd() + "/faiss")
```

Step 4 Create an application to serve the image analysis UI View details Open completed 1/25/2023 11:15 AM

```
    return jsonify(result)
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Application parameters')
    parser.add_argument('-p', '--port', dest='port', type=int,
                        help='port to run model',
default=os.environ.get("CDSW_READONLY_PORT"))

    args, unknown = parser.parse_known_args()
    port = args.port
    app.run(port=port)
```



Made with ♥ at Cloudera Fast Forward Labs.