plotly

Star 23,446

Dash Python **>** *Databricks Integration* **>** ***Executing Databricks Jobs using Plotly Dash***

> Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. **Sign up for early access now.**

Executing Databricks Jobs using Plotly Dash

> This page documents using the Databricks Python SDK to create and run Databricks jobs from within a Dash app. It is based on the following blog post: **Databricks SDK + Plotly Dash — the easiest way to get Jobs done**. We will be adding more content to the blog about Databricks in Dash in the near future.

The Databricks Jobs API allows you to manage and run Databricks compute jobs from outside of the Databricks environment. This allows you to productionize Databricks workflows by using the output of a Databricks job in a Dash app.

Here, we'll show you how to create a Dash app that takes user input via a dropdown and input field and passes that user input to a job on a Databricks cluster and then uses the job's output.

The Databricks Notebook

In our Dash app, we'll be passing parameters to a Databricks notebook on a Databricks cluster. For our example, we use a notebook named `my_notebook` that generates a forecast based on the input parameters and saves a figure that plots this forecast.

This notebook is the **notebook from the Databricks SDK + Plotly Dash blog post** and here we explain the main features of this notebook and why they are important for what happens in our Dash app.

Input Parameters

We want to use values selected in our Dash app when the Databricks notebook runs. This notebook has **input widgets** that allow it to accept parameters sent from the Dash app. In this example notebook, they are used to filter data in the Databricks notebook.

The widgets are defined at the top of the notebook. Note the names of these widgets, "us-state" and "forecast-forward-days", as we'll be using them in our Dash app.

```
# We will pass values to these from our Dash app
dbutils.widgets.text("us-state", "All States", "State Dropdown")
dbutils.widgets.text("forecast-forward-days", "180", "Forecast days")
```

They are then used to filter the data:

```
selected_state = dbutils.widgets.get("us-state")
if selected_state != "All States":
    cleaned_data = cleaned_data.filter(col("state") == selected_state)
```

And to set the number of days to make the prediction for:

```
num_days = int(dbutils.widgets.get("forecast-forward-days"))
```

When we run the job later, these widget values will be updated based on the values selected in the Dash app's UI.

The notebook then generates a forecast and displays it as a Plotly figure. How the notebook generates the forecast is not important for our Dash app, just that it generates an output that we can use. See the **notebook**

from the Databricks SDK + Plotly Dash blog post for the full code of generating a forecast with the input parameters.

Notebook Output

We want to use the output of the notebook to display a graph in our Dash app. In this notebook, we generate a Plotly figure that shows forecasts. At the end of the notebook, we then save that figure's data so it can be used in the Dash app. Note the location that we are saving the output to (the `path_to_save` variable), as we'll be using that in our Dash app.

```
import plotly.tools as tls
import json
import plotly.utils

# Here, we write the Plotly figure to JSON
fig_json = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

# We store the JSON in Databricks File Storage. We'll use this path in our Dash app to load the
path_to_save = "/tmp/forecast_plot.json"
dbutils.fs.put(path_to_save, fig_json, overwrite=True)
```

The Dash App

Let's create a Dash app that connects to a Databricks cluster, passes parameters to the notebook, runs the notebook, and uses its output.

Installation

The Dash app below uses the **Databricks SDK** to connect to the Databricks cluster. Install it with:

```
pip install databricks-sdk
```

Configuration

In the Dash code example below, we configure the Dash app to authenticate to Databricks and set the compute to use for the job using environment variables. To run the app and connect to Databricks, you'll need your Databricks Host, a token to authenticate to Databricks, and the ID of the cluster where the job will run.

- `DATABRICKS_HOST=<your-databricks-host>`

  The `DATABRICKS_HOST` is the URL where you log in to Databricks. For example, if you log in at `https://cust-success.cloud.databricks.com/`, that is the value to use for the `DATABRICKS_HOST` environment variable.

- `DATABRICKS_TOKEN=<your-databricks-token>`

  The `DATABRICKS_TOKEN` is a personal access token. For details on generating and managing personal access tokens, see **Databricks personal access token authentication in the Databricks docs**.

- `DATABRICKS_CLUSTER_ID=<your-cluster>`

  See **the Get identifiers for workspace objects page in the Databricks docs** for details on how to get your cluster ID.

Dash Enterprise

For an app deployed to Dash Enterprise or running in workspaces, add these environment variables **in your app's settings tab**.

Local Development

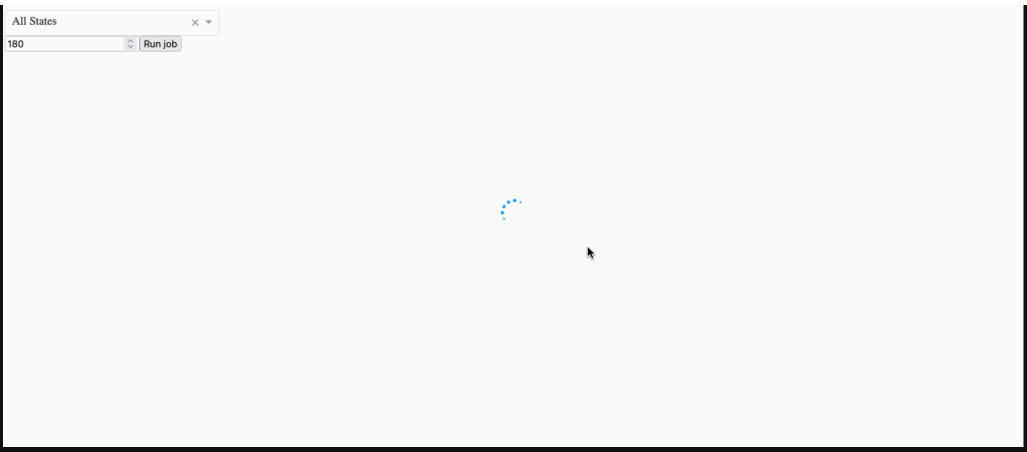
To run the following example when developing locally, pass the environment variable values when launching the app.

```
DATABRICKS_CLUSTER_ID=<your-cluster> DATABRICKS_HOST=<your-databricks-host> DATABRICKS_TOKEN=<y
```

Code

In the following Dash app, we create the job on Databricks and run it. The app has a dropdown, number input, and a button to run the job. It also has a callback which is triggered by the button. It's within this callback that we create and run the notebook using the Databricks SDK. We then use the output of the job to generate a graph that we display in the Dash app.

```
from dash import Dash, dcc, html, callback, Input, Output, State

from databricks.sdk import WorkspaceClient
from databricks.sdk.service import jobs
import time
import plotly.graph_objs as go
import base64
import json
import os


app = Dash(__name__)
server = app.server

us_states = [
    "All States", "AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL'
    "IA", "KS", "KY", "LA","ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", '
    "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA",
]

app.layout = html.Div([
                dcc.Dropdown(us_states, value="All States", id="state-dropdown", style={'widtl
                dcc.Input(value=180, type="number", id="forecast-forward-days"),
                html.Button("Run job", id="run-job"),
                dcc.Loading(
                    id="loading",
                    children=html.Div(id="forecast-plot", style={"height": 450}),
                    type="circle",
                )

    ])


@callback(
    Output("forecast-plot", "children"),
    State("state-dropdown", "value"),
    State("forecast-forward-days", "value"),
    Input("run-job", "n_clicks"),
    prevent_initial_call=True,
)
def run_job(state, forecast_days, n_clicks):
    # Create an instance of WorkspaceClient and authenticate using environment variables
    w = WorkspaceClient(
        host=os.environ["DATABRICKS_HOST"],
        token=os.environ["DATABRICKS_TOKEN"]
    )

    # Capture values from the Dash app to pass to the job in Databricks
    # Note how these match the widget names in the notebook
    params_from_dash = {"us-state": state, "forecast-forward-days": forecast_days}

    # Notebook location on Databricks
```



Notes on the above example:

- We import all the required libraries, including the parts of the Databricks SDK that we'll use: `from databricks.sdk import WorkspaceClient` and `from databricks.sdk.service import jobs`
- Our app layout has a `dcc.Input` and a `dcc.Dropdown` to accept user input and a `html.Button` to run the job.
- When the app user selects the **Run job** button, the values from the `dcc.Dropdown` and `dcc.Input` components are passed into the callback.
- We authenticate to Databricks using our environment variables:

```
w = WorkspaceClient(
        host=os.environ["DATABRICKS_HOST"],
        token=os.environ["DATABRICKS_TOKEN"]
)
```

- We capture the values from the `dcc.Dropdown` and `dcc.Input` that will be passed to our Databricks notebook. This a `dict`, where the keys represent the names of the widgets in the Databricks notebook and the values are the names of the values within our Dash callback: `params_from_dash = {"us-state": state, "forecast-forward-days": forecast_days}`
- We define the location of our notebook on Databricks in our `notebook_path` variable.
- Next, we try to connect to the cluster we are going to use (set in an environment variable), starting it if it is not already running.
- We create a job, `created_job`, with one task: the task to run the notebook. To the `jobs.Task` instance, we pass the cluster ID, and details about the notebook, including the parameters.
- With the job created, we run it: `w.jobs.run_now(job_id=created_job.job_id).result()`
- When the job completes (when the notebook finishes running), it saves the output and we then read it into the Dash app with `fig_bytes = w.dbfs.read("/tmp/forecast_plot.json")`.
- We process the data from the job output, and use it in a Plotly Figure, `fig = go.Figure(fig_data)`.
- This figure is then rendered in a `dcc.Graph` component which is our callback's output.

Dash Python **>** *Databricks Integration* **>** ***Executing Databricks Jobs using Plotly Dash***