BERKELEY LAB

Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

Dash: A Python Framework for Building Web Applications

Evan Mladina, Karol Krizka, Timon Heim, Carl Haber

Student Instrumentation Meeting -- May 8, 2020

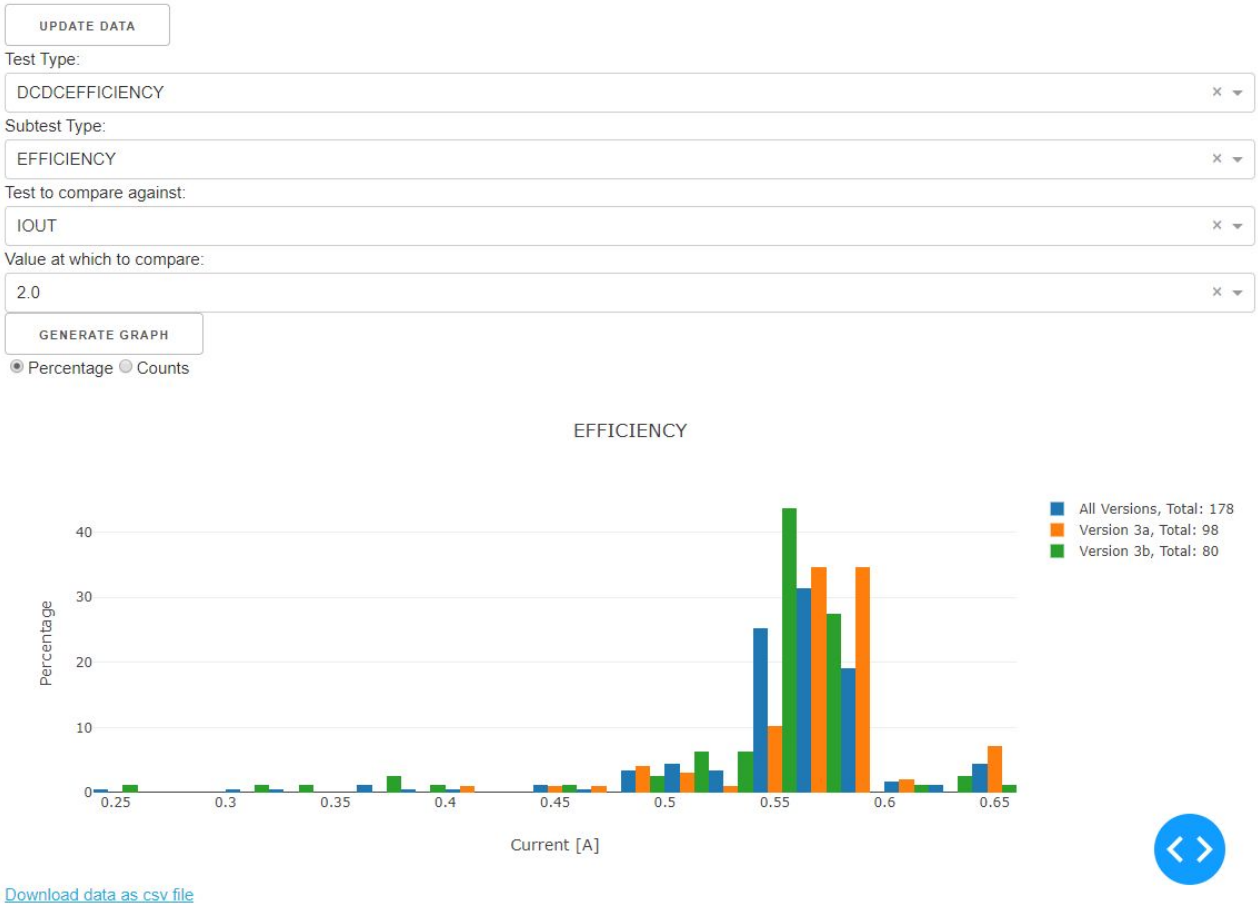
What is Dash

- Dash is a python module used for building web applications.
  - Written on top of Flask, Plotly.js, and React.js
- Primary usage is for building data visualization apps.
- Can be extended to do more...

Handy Dash References

- **Dash tutorial:**
  - https://dash.plotly.com/installation
- **Example dash apps:**
  - https://dash-gallery.plotly.host/Portal/
- **Dash component library:**
  - https://dash.plotly.com/dash-core-components

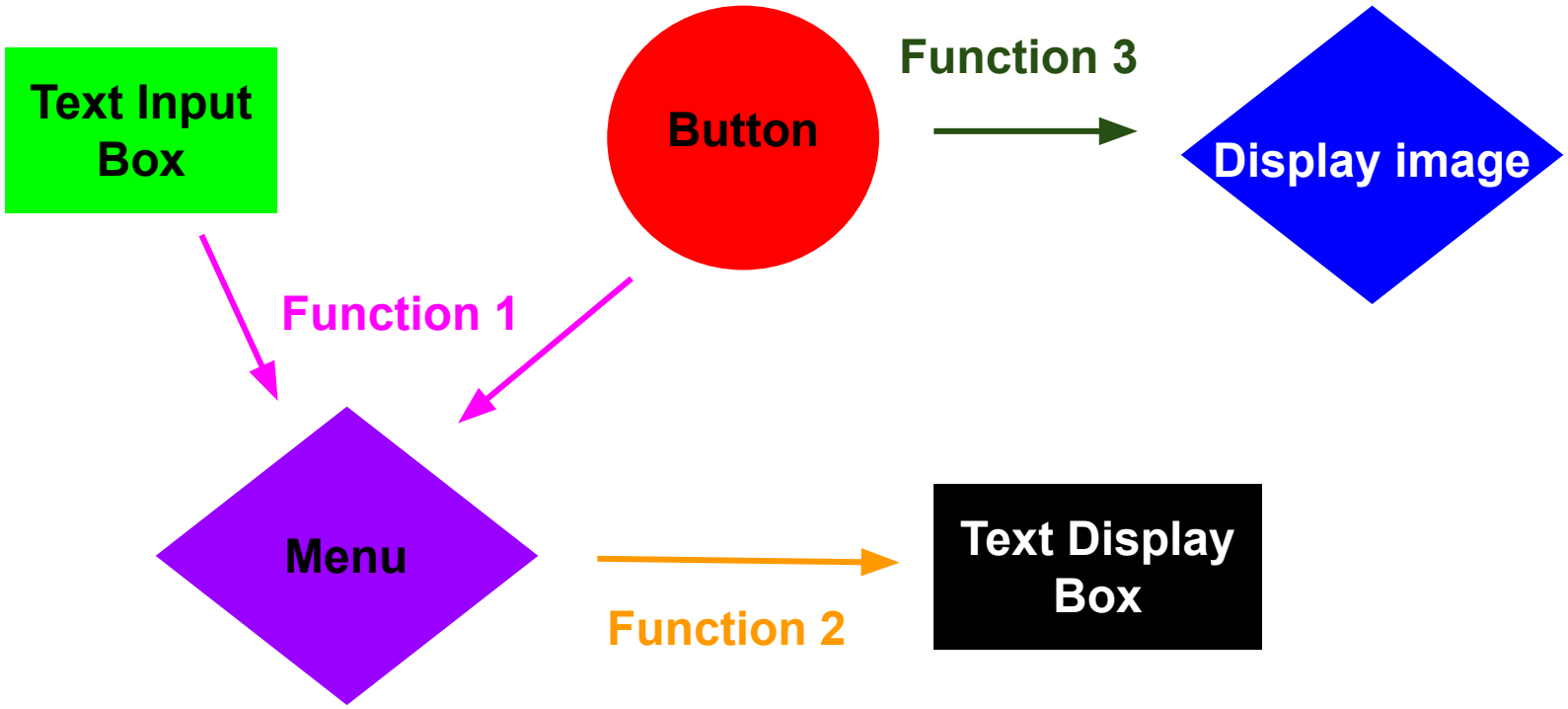
ITk Database Histogram App



- To view the webpage run the following terminal command and go to 127.0.0.1:8050 on your local browser.
  - ssh -L 8050:localhost:8050 username@seadog.dhcp.lbl.gov
- The page will soon be available publicly at itk-strips-powerboard.lbl.gov

How does it work?

- **Essential Idea: Web page consists of a collection of components and collection of functions that are triggered whenever an aspect of a given component changes.**



- **Populate webpage with various dash core components or html components.**

```
dcc.Dropdown(
    id='test-type',
    options=[{'label': i, 'value': i} for i in testResults.keys()],
    value='DCDCEFFICIENCY'
```

```
dcc.Input(id='version', placeholder='Powerboard Version Number')
```

```
dcc.Checklist(id='pb-2', options = [
    {'label': 'Board 2', 'value': '2'}],
    value = ['2']),
```

Dash core components

- dcc.Checklist
- dcc.ConfirmDialog
- dcc.ConfirmDialogProvider
- dcc.DatePickerRange
- dcc.DatePickerSingle
- dcc.Dropdown
- dcc.Graph
- dcc.Input
- dcc.Interval
- dcc.Link
- dcc.Loading
- dcc.Location
- dcc.LogoutButton
- dcc.Markdown
- dcc.RadioItems
- dcc.RangeSlider
- dcc.Slider
- dcc.Store
- dcc.Tab
- dcc.Tabs
- dcc.Textarea
- dcc.Upload

- **Define functions called callbacks that are triggered when an aspect of a given component changes.**

```
@app.callback(
    [dash.dependencies.Output('subtest-type', 'options'),
     dash.dependencies.Output('subtest-compare', 'options')],
    [dash.dependencies.Input('test-type', 'value')])
def update_subtests(testType):
    subtest_type=[{'label': i, 'value': i} for i in testResults[testType]]
    subtest_compare=[{'label': i, 'value': i} for i in testResults[testType]]
    return subtest_type, subtest_compare
```

- **When the 'value' property is changed in the component with id 'test-type', the 'options' for both 'subtest-type' and 'subtest-compare' are changed**

```
dcc.Dropdown(
    id='test-type',
    options=[{'label': i, 'value': i} for i in testResults.keys()],
    value='DCDCEFFICIENCY'
```

Input

```
dcc.Dropdown(
    id='subtest-type',
    options=[{'label': i, 'value': i} for i in testResults['DCDCEFFICIENCY'].keys()],
    value='IOUT'
```

Output

ITk Database Histogram App

UPDATE DATA

Changing this

Test Type:

DCDCEFFICIENCY

Subtest Type:

EFFICIENCY

Changes these

Test to compare against:

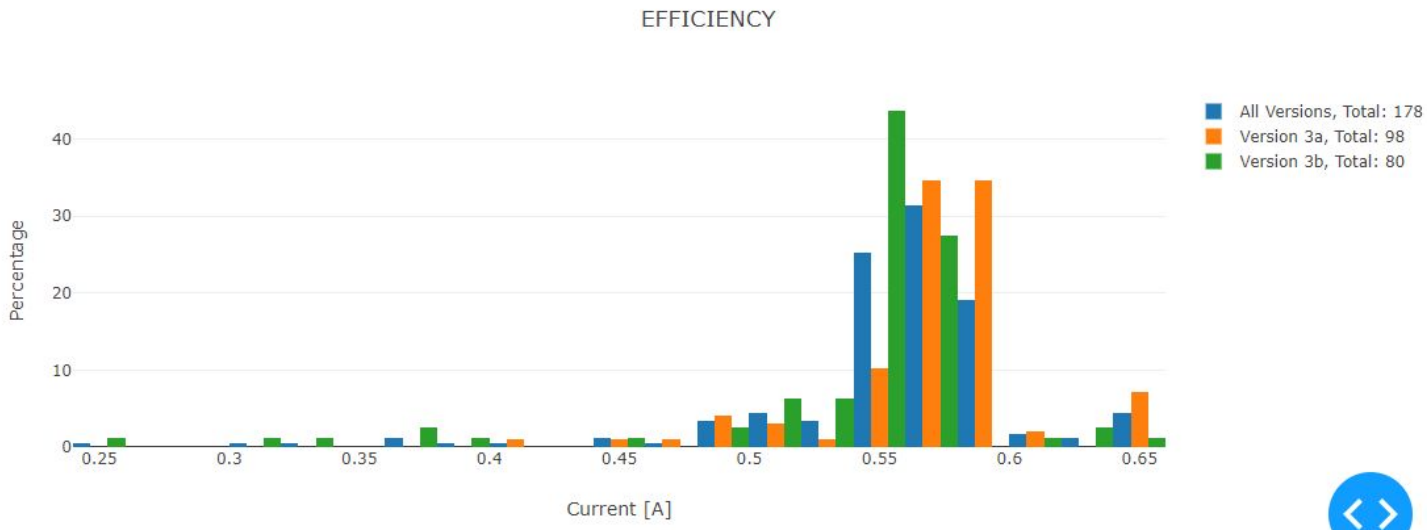
IOUT

Value at which to compare:

2.0

GENERATE GRAPH

Percentage Counts

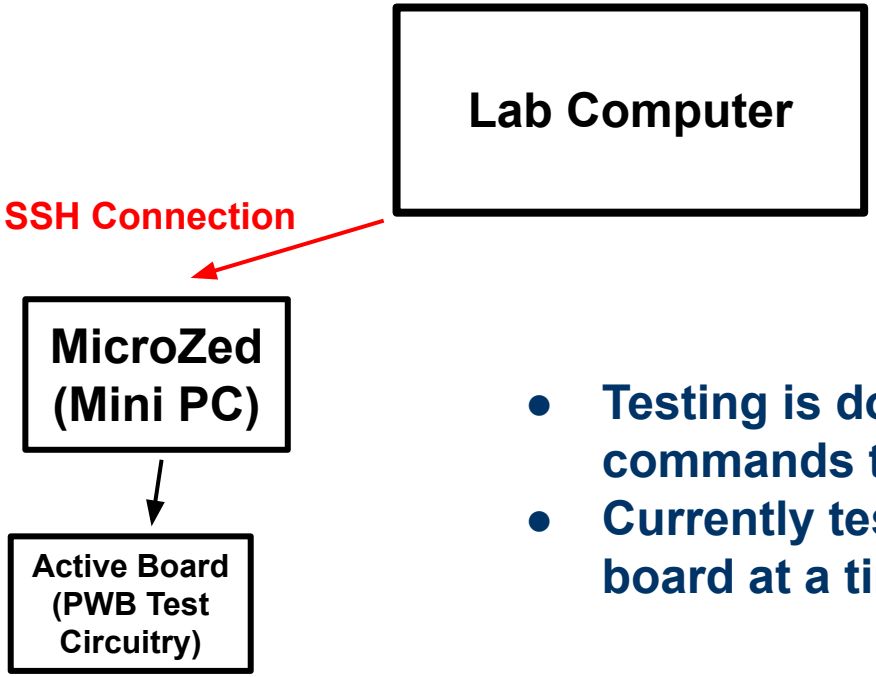


Download data as csv file

- Everytime you change an input that is used in a callback, that callback will fire.
- Can define callbacks with multiple inputs.
  - Changing any one of those inputs will trigger the callback.
- Can use States instead, and a button to trigger the callback.

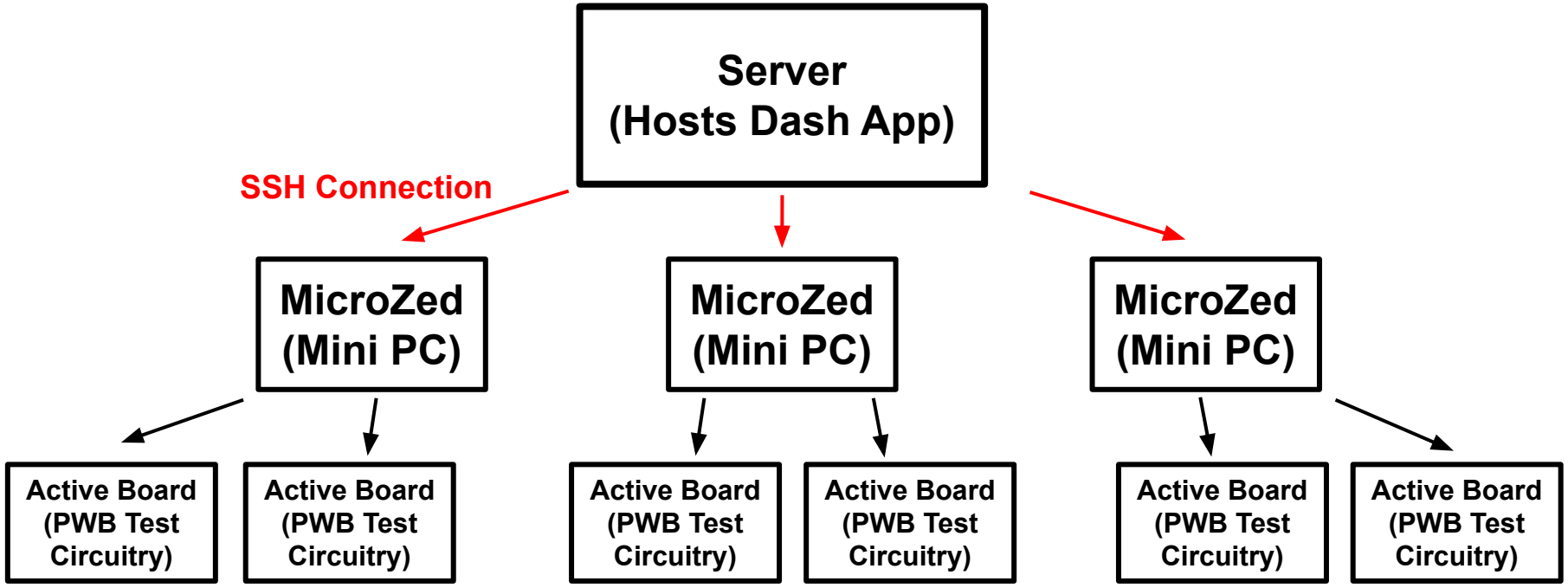
```
@app.callback(
    Output('display-diagnostic-test', 'children'),
    [Input('diagnostic-test-button', 'n_clicks')],
    [State('panel-number', 'value'),
     State('version', 'value'),
     State('batch', 'value'),
     State('pb-0', 'value'),
     State('pb-1', 'value'),
     State('pb-2', 'value'),
     State('pb-3', 'value'),
     State('pb-4', 'value'),
     State('pb-5', 'value'),
     State('pb-6', 'value'),
     State('pb-7', 'value'),
     State('pb-8', 'value'),
     State('pb-9', 'value'),
     State('serial-0', 'value'),
     State('serial-1', 'value'),
     State('serial-2', 'value'),
     State('serial-3', 'value'),
     State('serial-4', 'value'),
     State('serial-5', 'value'),
     State('serial-6', 'value'),
     State('serial-7', 'value'),
     State('serial-8', 'value'),
     State('serial-9', 'value'),
     State('debug', 'value')]
)
```

Current Powerboard Testing Procedure



- Testing is done by running commands through the terminal.
- Currently testing on one active board at a time.

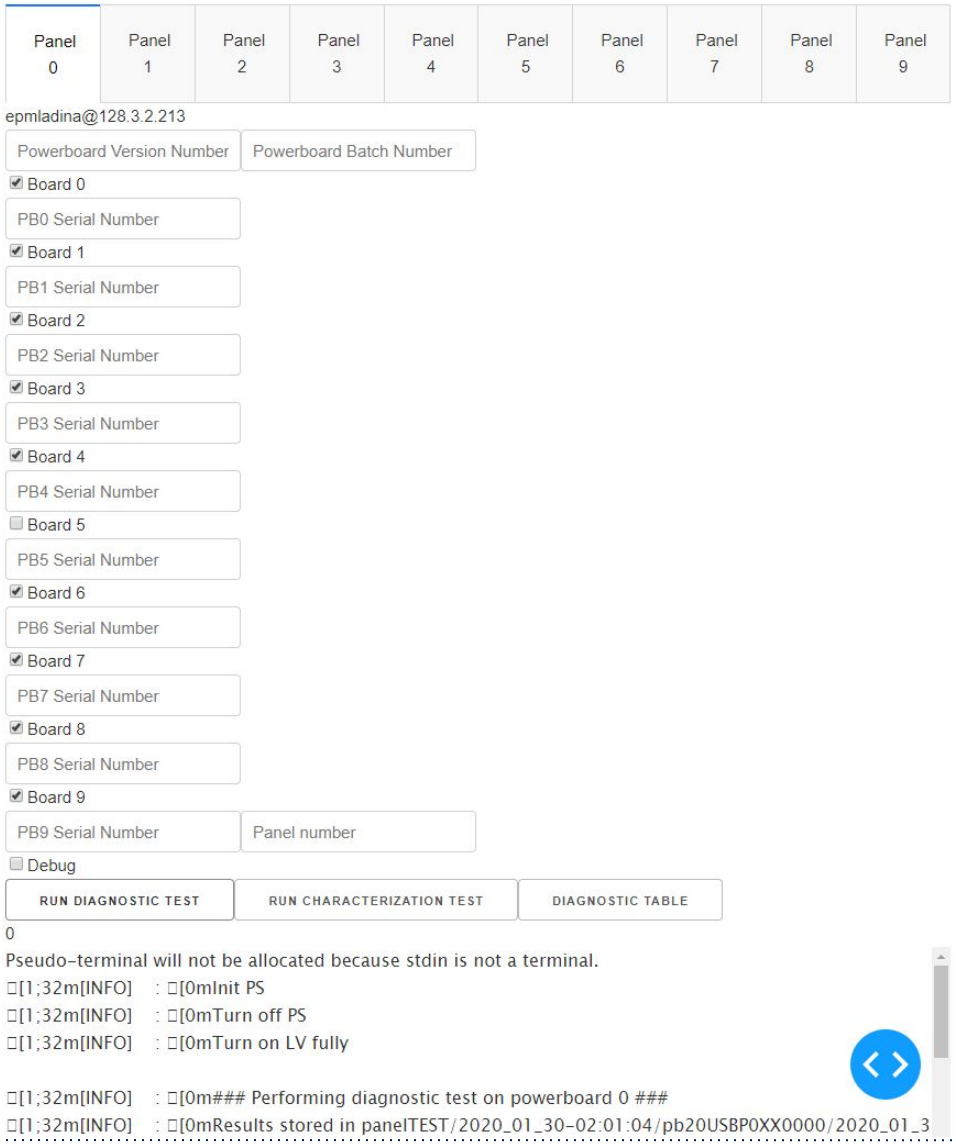
Planned Powerboard Testing Procedure



- Dash app spins off subprocesses that SSH into the MicroZeds and run the tests.
- Network file system allows for sharing of data across all computers.
- All done by button clicking and parameter selection in dash app.

Dash Testing App

- Tabs are used to run tests on different panels.
- Buttons ‘Run Diagnostic Test’ and ‘Run Characterization Test’ spin off subprocesses run on a MicroZed.
- Terminal out is saved to a file and displayed on the webpage.



- **Dash runs on Flask, a python framework for hosting python based web applications.**

```
-bash-4.2$ ./dashPWBApp.py
Running on http://127.0.0.1:8050/
Debugger PIN: 413-693-371
 * Serving Flask app "dashPWBApp" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
Running on http://127.0.0.1:8050/
Debugger PIN: 489-974-335
```

- **Running your dash app this way, the site will only be accessible through the machine you are running it on.**
- **Extra steps are needed to host this publicly.**

Hosting Your Web Application Publicly

Required Dependencies

In order to get going you will need the following:

- **Apache (httpd on CentOS)**
- **Python3**

Python packages:

- **Flask**
- **Dash**
- **mod_wsgi**
- **Virtualenv**

**** Make sure all dependencies are compatible with one another other!**

The following tutorial for the most part follows the tutorial given here:

https://dev.to/sm0ke/flask-deploy-with-apache-on-centos-minimal-setup-2kb7

For that reason, I will skip some of the initial setup as it is outlined in the link above, however, the tutorial given in the link does not show you how to properly integrate Dash, so we will diverge from the tutorial in what goes into the following files:

- **run.py**
- **app/__init__.py**
- **wsgi.py**

We will also not be editing /etc/httpd/conf/httpd.conf, but will instead be creating a new file that contains the VirtualHost definition.

Modifications to app/__init__.py

Following the linked tutorial, go ahead and create app/__init__.py, but leave out the function definition and keep the following:

```
from flask import Flask
app = Flask(__name__)
```

As the typical name for your dash application is “app”, in the above line we want to change “app” to “server”, like so:

```
server = Flask(__name__)
```

Modifications to app/__init__.py

Now, app/__init__.py is where we want to put our Dash app, so go ahead and move it there. If you have created your Dash app already you'll note that at the begin of your python file you have the declaration of your application:

```
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
```

Change it to the following:

```
app = dash.Dash(__name__, server=server, routes_pathname_prefix='/', external_stylesheets=external_stylesheets)
```

**Note: If you leave the function definition as given in the tutorial, and changing app.route to server.route and routes_pathname_prefix to '/extention_of_your_choosing/' you could have a homepage displaying the function output, and your dash app would be located at server_domain/extention_of_your_choosing.

Modifications to app/__init__.py

If you built your dash app following the dash tutorial, you should have the following at the bottom of the file:

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

Go ahead and remove this, as it will now be moved to run.py

Modifications to run.py

run.py doesn't look much different from what is outlined in the linked tutorial, again we just have to change "app" to "server". So, run.py should look like this:

```
import os
from app import server

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    server.run(host="0.0.0.0", port=port, debug=True)
```

Modifications to wsgi.py

We also have to modify wsgi.py.

Copy wsgi.py exactly as is outlined in the tutorial, changing the following:

```
from app import app as application
```

to:

```
from app import app
application = app.server
```

Apache Modifications

Do not edit /etc/httpd/conf/httpd.conf and instead create file under /etc/httpd/conf.d and call it what you wish (with a .conf extension). In our case it is titled pwbDashPlotswsgi.conf

The file should look like so:

```
<VirtualHost *:80>

    ServerName itk-strips-powerboard.lbl.gov

    WSGIDaemonProcess pwbDashPlots python-home=/var/www/pwbDashPlots

    WSGIScriptAlias / /var/www/pwbDashPlots/wsgi.py

    <Directory /var/www/pwbDashPlots>
        Require all granted
    </Directory>

</VirtualHost>
```

Replace itk-strips-powerboard.lbl.gov with your server name.

Error Checking

If you encounter any errors with displaying your webpage, check the error log by opening up /var/log/httpd/error_log.

The log will be written to every time apache is restarted, you run the flask app, or you try to access the webpage, so check it after each step when debugging.