plotly

Star 23,447

Dash Python > ***Embed Your Dash App in Other Websites***

Embedding Dash Apps in Other Websites

You may want your Dash apps to be used within existing websites instead of as standalone apps. With this strategy, the Dash app is *embedded* into another web app, referred to as the *host* or *parent* app.

This page provides an overview of two solutions for embedding Dash apps: Dash Enterprise **Embedding Middleware** and iframes.

Dash Enterprise can be installed on the cloud services of **AWS**, **Azure**, or **Google**.

Dash Enterprise Embedding Middleware

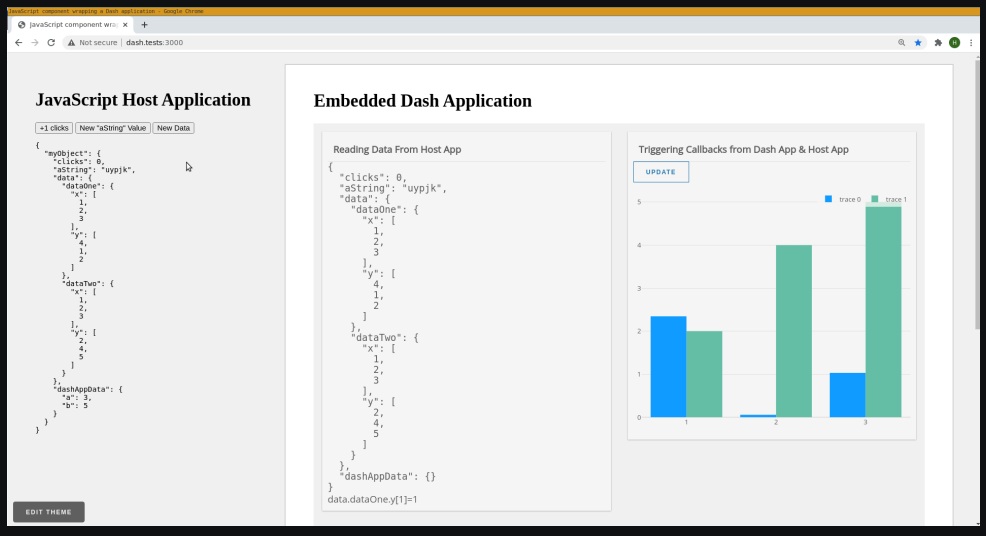
To get started with Dash Enterprise Embedding Middleware, look up the documentation that is included with Dash Enterprise. The URL for the documentation depends on the hostname that your administrator has chosen. **Find out if your company is using Dash Enterprise**.

Dash Enterprise Embedding Middleware is one of the powerful libraries that you gain access to when your organization licenses Dash Enterprise. It's a first-class capability for embedding Dash apps as a microservice in websites and web platforms that are external to Dash Enterprise.

Dash Enterprise Embedding Middleware is our recommended solution for securely embedding production-grade Dash apps. It has the following advantages:

- **Bidirectional data flow**: Dash Enterprise Embedding Middleware allows the host app and embedded app to work together as parts of a whole. For a unified user experience, you may want interactions in the embedded Dash app to result in changes to the host app. The host app and embedded Dash app can exchange data this way in both directions, sharing state and allowing you to treat them as a single app during development.

  In the example below, data is passed from a host JavaScript app to the embedded Dash app and vice versa.



- **Single sign-on (SSO)**: If the website that you want to embed your Dash app into requires users to log in, they don't have to log in again to view the embedded Dash app. Dash Enterprise Embedding Middleware leverages your existing website's authentication logic so that only users who have logged into the host web

application can view the embedded Dash app. This means that you don't have to rewrite any authentication code and the embedded Dash app benefits from the password protection that is already in place.

- **Salesforce integration**: Dash Enterprise Embedding Middleware supports embedding Dash apps into Salesforce. The embedded apps use the Salesforce API to ensure that users only have access if they are logged into Salesforce. In-depth guides and examples for integrating with Salesforce are provided in the documentation included with Dash Enterprise.

Implementation Overview

Embedding your Dash app using Dash Enterprise Embedding Middleware involves using two Plotly-provided packages:

- Dash Embedded Component (`dash-embedded-component`): Loaded or imported in the host application.
- Dash Embedded (`dash-embedded`): Used within the Dash app that you want to embed. Comes with the `Embeddable` plugin.

Here's what your code might look like to set up Dash Enterprise Embedding Middleware and take advantage of state sharing between the host and embedded apps.

If you're using a JavaScript host app, you load Dash Embedded Component. Then, you provide the array or object that you want to share to your Dash app in the Dash Embedded Component `renderDash()` function:

```
<head>
    ...
    <script type="text/javascript" src=".../dash-embedded-component.js"></script>
</head>

...

var setter = window.dash_embedded_component.renderDash(
    { url_base_pathname: "http://dash.tests:8050" },
    'dash-app',
    sharedData
);
```

If you're using a React host app, the pattern is similar. You import Dash Embedded Component and use it inside JSX:

```
import { DashApp } from "dash-embedded-component";

window.React = React;
window.ReactDOM = ReactDOM;
window.PropTypes = PropTypes;

class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            sharedData: {
                myObject: {
                    clicks: 0,
                    aString: randomString(5),
                    data: myData,
                    multiplyFunc: (x, y) => { ... },
                    sumFunc: (x, y) => { ... },
                    storeDataFromDash: obj => { ... },
                    dashAppData: {}
                },
            },
        };
        this.clickIncrement = this.clickIncrement.bind(this);
        ...
    }

    render() {
        return (
        <div className="App-Background">
            ...
            <div className="App-Content">
```

```
            <h1>Embedded Dash app</h1>
            <DashApp config={url_base_pathname: "http://dash.tests:8050"} value={this.state.sha
            </div>
        </div>
        );
    }
}
```

Inside the Dash app that you want to embed, you import `dash_embedded` and the `Embeddable` plugin. You register the plugin and use its `origin` argument to allow the host application to access the embedded app.

To consume the shared data, you use `dash_embedded.ConsumerContext` and `dash_embedded.ConsumerFunction`.

```
from dash import Dash
from dash_enterprise_libraries import ddk
from dash_enterprise_libraries import dash_embedded
from dash_embedded import Embeddable

app = Dash(__name__, plugins=[Embeddable(origins=['https://<your-host-application-hostname>']

...

app.layout = ddk.App(
    [
        ...
        ddk.Card(
            [
                ddk.CardHeader(title="Triggering callbacks from Dash app & host app"),
                dash_embedded.ConsumerContext(id="clicks", path=["myObject", "clicks"]),
                dash_embedded.ConsumerContext(id="data-one", path=["myObject", "data", "dataOr
                dash_embedded.ConsumerContext(id="data-two", path=["myObject", "data", "dataTv
                ...
            ],
            width=50,
        ),
        ddk.ControlCard(
            [
                ddk.CardHeader(title="Triggering host app functions from Dash app"),
                dash_embedded.ConsumerFunction(
                    id="host-app-multiply", path=["myObject", "multiplyFunc"]
                ),
                dash_embedded.ConsumerFunction(id="host-app-sum", path=["myObject", "sumFunc"]
                ddk.ControlItem(...),
                ...
            ],
            width=50,
        ),
        ...
    ]
)
...

# Access the nested objects via `path=["myObject"]`
@callback(Output("display-full-object", "children"), Input("full-object", "value"))
def display(value):
    return json.dumps(value, indent=2)


# Access nested values via `path=[...]`
@callback(
    Output("display-data_dataOne_y[1]", "children"), Input("data_dataOne_y[1]", "value"))
def display(value):
    return "data.dataOne.y[1]={}".format(value)
```

The exact implementation depends on your Dash Enterprise version and the host application stack. In-depth guides and examples are provided in the documentation included with Dash Enterprise. The documentation covers embedding inside host apps written in JavaScript, React, React with Webpack, Vue, and AngularJS.

Dash Enterprise Embedding Middleware cannot be used to embed data apps that are written in non-Dash frameworks.

Embedding Apps Using <iframe>

A basic approach to embedding a data app in an existing web application is to include an `<iframe>` element in your HTML whose `src` attribute points to the address of a deployed app. This allows you to embed your app in a specific location within an existing web page using your desired dimensions:

```
<iframe src="http://localhost:8050" width=700 height=600>
```

Note that this solution has several limitations:

- The embedded app must be public and cannot integrate with any existing authentication system in the host application.
- It can be difficult to implement data sharing, especially if the embedded app needs to write back to the host application.
- Iframes can be slow, load asynchronously, or have poor mobile responsiveness.
- Iframes are not allowed in some organizations.

Dash Python ***> Embed Your Dash App in Other Websites***