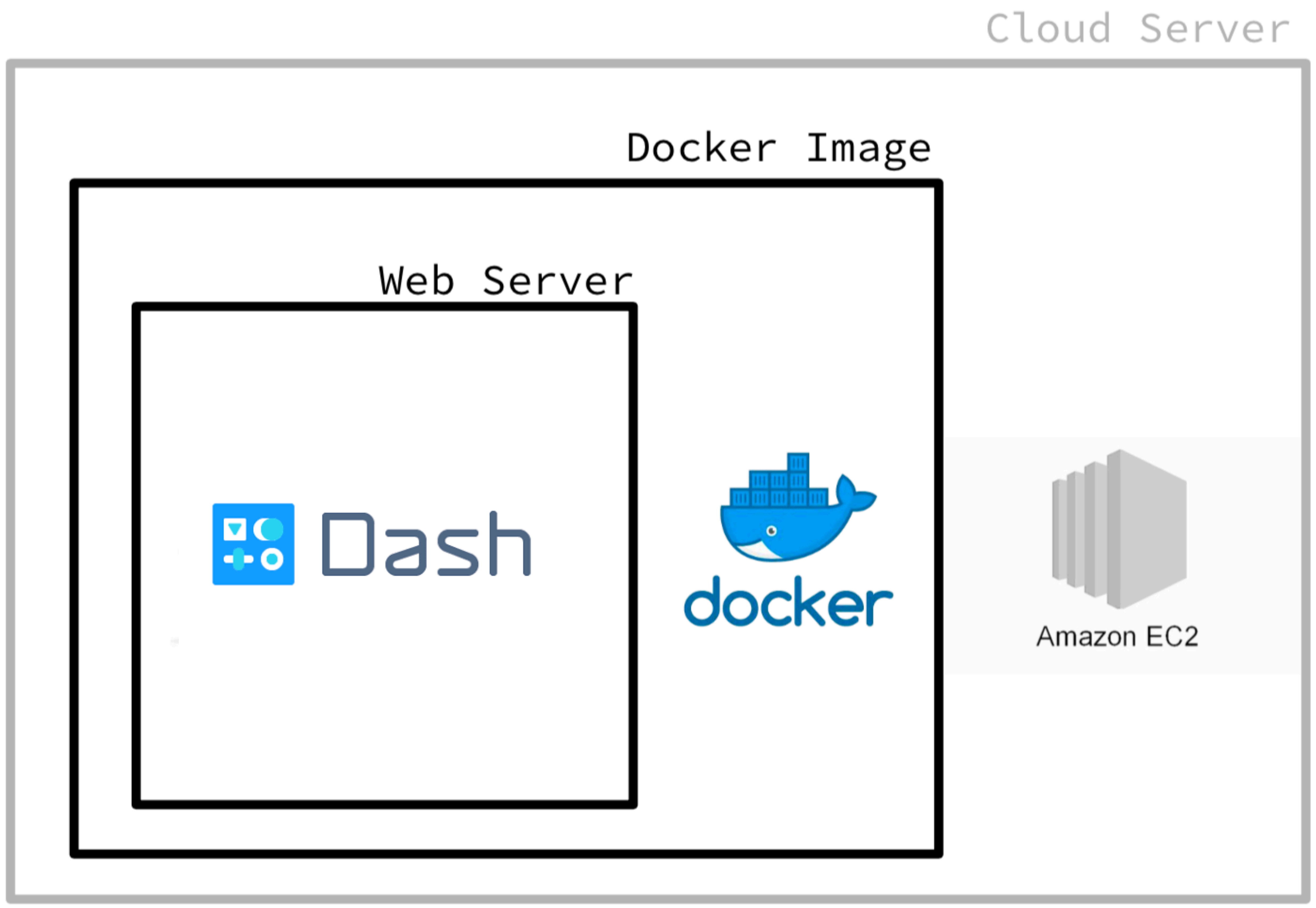
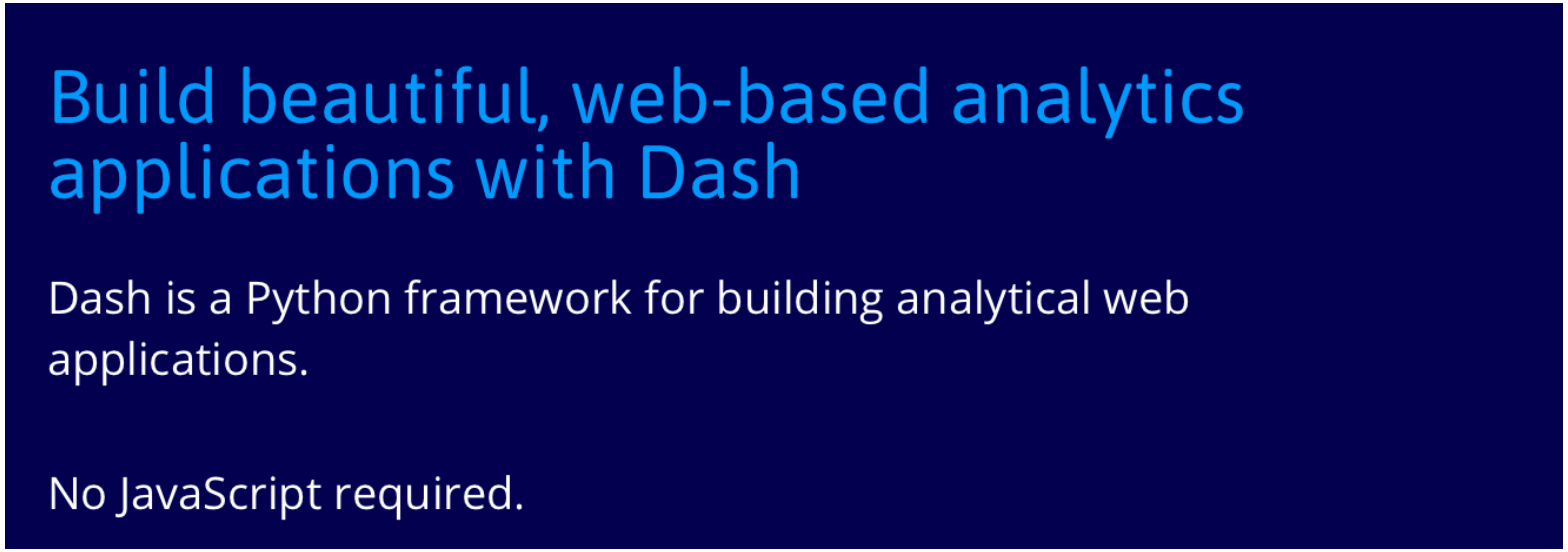
What are we going to build?



Our final tool of the course is Dash

What is Dash?



Build beautiful, web-based analytics applications with Dash

Dash is a Python framework for building analytical web applications.

No JavaScript required.

https://plot.ly/products/dash/

Dash apps are composed of two parts

The `app.layout` which of course is the ‘layout’ of the app to describe what the application looks like

The second piece is the `app.callback` which describes the interactivity of the application `app.layout`

Provides Python classes for all of the visual components of the application.

Most components come from either the `dash_core_components` and `dash_html_components`

For those so inclined can build their own classes using JavaScript

Structure of `app.layout`

The `layout` is composed of a tree of "components" like `html.Div` and `dcc.Graph`

The dash_html_components library has a component for every HTML tag. The `html.H1(children ='Hello Dash')` component generates a `<h1>Hello Dash</h1>` HTML element in your application

Not all components are pure HTML.

Each component is described entirely through keyword attributes: you will primarily describe your application through these attributes

`children` is always the first attribute which means that you can omit it: `html.H1(children ='Hello Dash')` is the same as `html.H1('Hello Dash')`

`Taken from https://dash.plot.ly/getting-started`

Reusable Components of `app.layout`

Can write markup in Python to create complex reusable components like tables without switching contexts or languages

We will see an example when creating a table in our iris example….

`Taken from https://dash.plot.ly/getting-started`

Markdown

Although Dash exposes HTML through the `dash_html_components` library, it can be tedious to write your copy in HTML. For writing blocks of text, you can use the `Markdown` component in the `dash_core_components` library

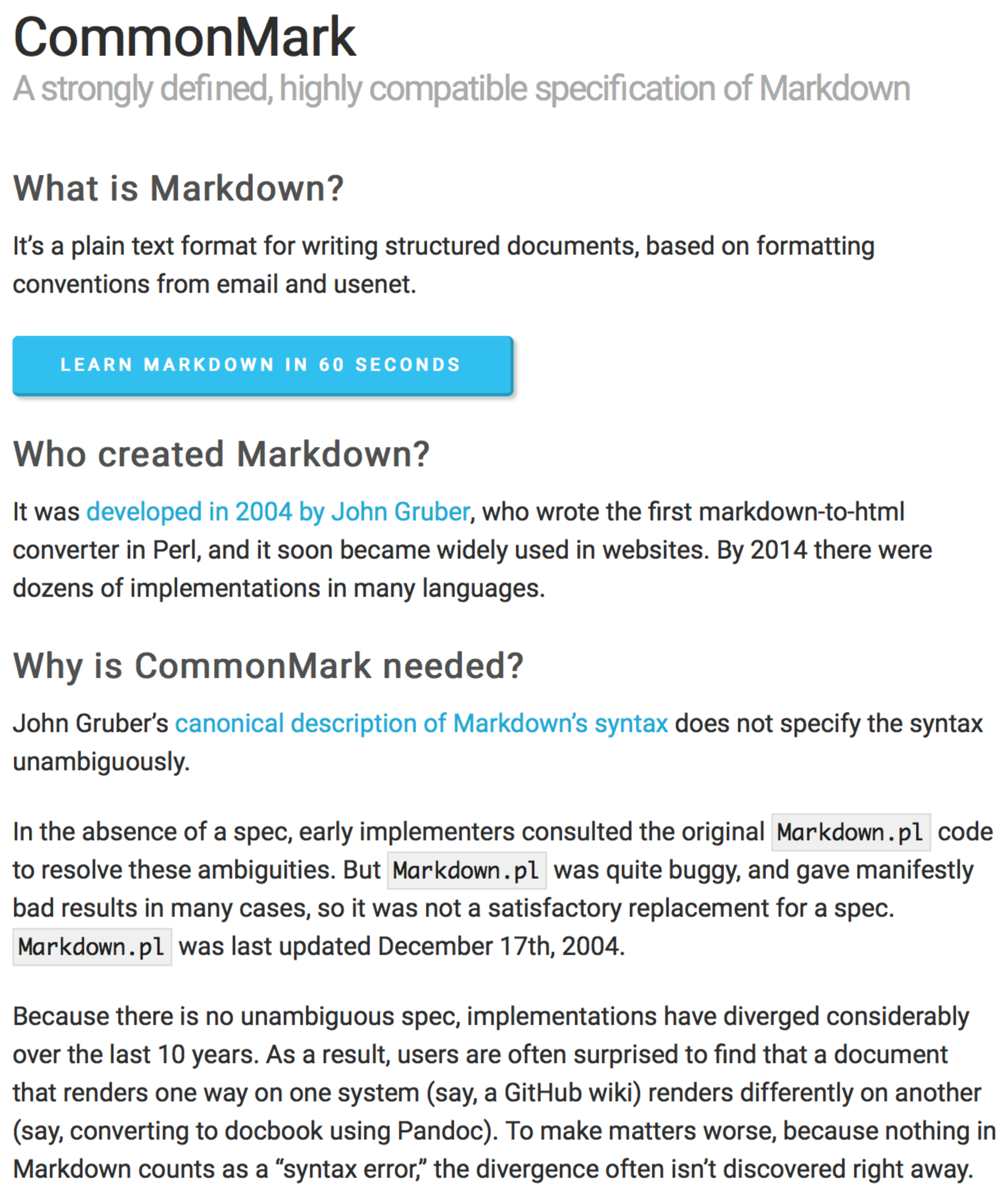
```
markdown_text = '''
### Dash and Markdown

Dash apps can be written in Markdown.
Dash uses the [CommonMark](http://commonmark.org/)
specification of Markdown.
Check out their [60 Second Markdown Tutorial](http://commonmark.org/help/)
if this is your first introduction to Markdown!
'''

app.layout = html.Div([
    dcc.Markdown(children=markdown_text)
])
```

Taken from https://dash.plot.ly/getting-started

Maybe one more tool…. Markdown

We've already seen this in action in our Github README's as well as a version of Rmarkdowns

Take a look at the 60-second tutorial and the 10-minute one as well



CommonMark

A strongly defined, highly compatible specification of Markdown

What is Markdown?

It's a plain text format for writing structured documents, based on formatting conventions from email and usenet.

Who created Markdown?

It was developed in 2004 by John Gruber, who wrote the first markdown-to-html converter in Perl, and it soon became widely used in websites. By 2014 there were dozens of implementations in many languages.

Why is CommonMark needed?

John Gruber's canonical description of Markdown's syntax does not specify the syntax unambiguously.

In the absence of a spec, early implementers consulted the original `Markdown.pl` code to resolve these ambiguities. But `Markdown.pl` was quite buggy, and gave manifestly bad results in many cases, so it was not a satisfactory replacement for a spec. `Markdown.pl` was last updated December 17th, 2004.

Because there is no unambiguous spec, implementations have diverged considerably over the last 10 years. As a result, users are often surprised to find that a document that renders one way on one system (say, a GitHub wiki) renders differently on another (say, converting to docbook using Pandoc). To make matters worse, because nothing in Markdown counts as a "syntax error," the divergence often isn't discovered right away.

`app.callback`

The The "inputs" and "outputs" of our application interface are described declaratively through the `app.callback` decorator

the inputs and outputs of our application are simply the properties of a particular component.

Whenever an input property changes, the function that the callback decorator wraps will get called automatically. Dash provides the function with the new value of the input property as an input argument and Dash updates the property of the output component with whatever was returned by the function.

`Taken from https://dash.plot.ly/getting-started`

Brief Summary

Dash apps are built off of a set of simple but powerful principles: declarative UIs that are customizable through reactive and functional Python callbacks.

Every element attribute of the declarative components can be updated through a callback and a subset of the attributes, like the `value` properties of the `doc.Dropdown`, are editable by the user in the interface.

`Taken from https://dash.plot.ly/getting-started`

Again more value is probably derived from following the tutorials yourselves on the Plotly website

OR

building a deployed version of one ourselves, lets take a look at the dash-app-418 repo and then deploy it in an AMI

This ends our survey of a variety of different Data Science Tools from data collection all the way through deployment. We saw a lot of stuff in 10 weeks!

Where to go from here?

K + TensorFlow

PyTorch + fastai

H_2O.ai

Spend more time in building advanced models to be deployed

Where to go from here?

NGINX

gunicorn

Work towards load balancing, security, and other considerations in productionalization