Getting Started with Meteor

Hey there!
I'm Patrick Coffey.

Twitter & Github: @patrickocoffeyo

Lets talk about Meteor!

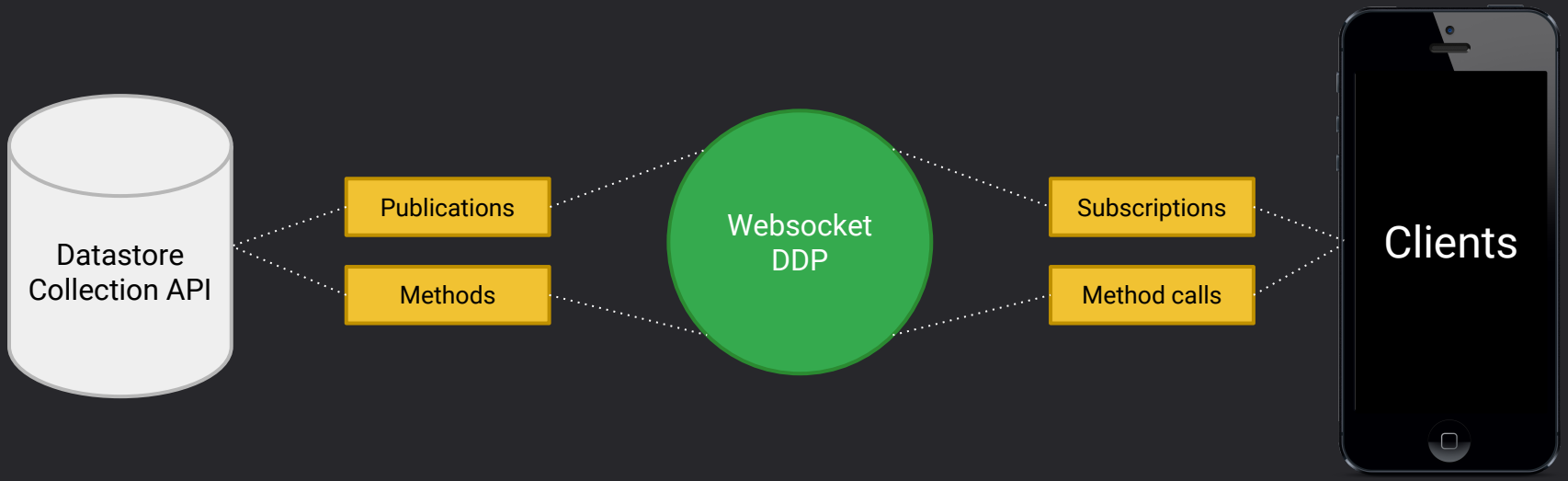
What is Meteor?

JS framework that enables developers to quickly create reactive, highly accessible applications.

What does Meteor do?

Stands between your server and client, and facilitates the organized transfer of data between the two.

What is Meteor?



Filesystem

Meteor publishes code to your client and/or server.

Your application will be naturally isomorphic.

Filesystem

Meteor knows where to publish code depending on where you put your files in your app.

Meteor respects a few folders, and publishes code to environments accordingly.

Filesystem

/*

Files that are in the root of the app, or in an unrecognized folder name get published to both the client and the server.

Filesystem

/lib

Files that are in this folder are published to both the client and server, but are loaded before any other files.

Good for code that needs to run first.

Filesystem

/client

Files within this folder get published to the browser, or any client.

Filesystem

/server

Files within this folder are run on the server only.

Filesystem

/public

Media files (images) within in this folder can be served to the client.

Files are accessible from the root path:

/public/logo.png = myapp.com/logo.png

Filesystem

/private

Media files within this folder can be used by the server using the Assets API.

The File System

Now that this is cleared up, let's get to the fun stuff!

How does Meteor actually work?

DDP

Distributed **D**ata **P**rotocol

Protocol for fetching structured data from a server, and receiving live updates when that data changes.

REST for websockets.

DDP

In a Meteor app, the client speaks to the server over websockets using the DDP protocol.

Client doesn't know/care if it's talking to Meteor server, just knows it's using DDP.

Collection API

JavaScript implementation of DDP, for JS clients and servers.

Collection API

Not specific to any database. Use MongoDB, Redis, as an example.

Collection API implementation doesn't change.

Collection API

Isomorphic. Run queries on the client or the server, and you'll get results.

Collection API

Creating a (mongodb) collection:

```
MyDocuments = new Mongo.Collection("mydocuments");
```

Collection API

Inserting data (client or server):

```
MyDocuments.insert({
  title: "A New Document",
  author: "Patrick Coffey",
  body: "This is my document body."
});
```

Collection API

Updating a data object (client or server):

```
MyDocuments.update({
  title: "A New Document"
}, {
  $set: {
    body: "Penguins are cool. Tux is my favorite!"
  }
});
```

Collection API

Deleting (client or server):

```
MyDocuments.delete({
  title: "A New Document"
});
```

Collection API

Meteor can sync these actions between the client and the server.

So… how does that work?

Publish/Subscribe API.

Meteor provides a mechanism that syncs data between the server and the client.

Publish/Subscribe API.

Meteor allows you to define datasets that it should sync by providing a pubsub API.

Publish/Subscribe API.

The server can define publications:

```
// Publishes documents by author.
if (Meteor.isServer) {
  Meteor.publish("MyDocumentsByAuthor", function (authorName) {
    // Make sure argument is a string.
    check(authorName, String);

    // Return a cursor for the MyDocuments collection.
    return MyDocuments.find({
      author: authorName
    });
  });
}
```

Publish/Subscribe API.

Client subscribes to publications, and uses documents:

```
// On the client, subscribe to "MyDocumentsByAuthor" for the author "Patrick Coffey".
if (Meteor.isClient) {
  Meteor.subscribe("MyDocumentsByAuthor", "Patrick Coffey").ready(function() {
    console.log('yay, subscription has been formed and data is available!');
  }));
}
```

Publish/Subscribe API.

Publications and subscriptions are reactive.. documents added to client/server are synced.

Permission system

Allows one to effectively define who can insert/update/remove documents.

```
MyDocuments.allow({
  insert: function (userId, document) { // return true || false; }
  update: function (userId, document) { // return true || false; }
  remove: function (userId, document) { // return true || false; }
});
```

Reactive template system

Easily create templates that use documents from a subscription:

Reactive template system

```
// In your client js.
if (Meteor.isClient) {
  Template.myTemplateName.helpers({
    myDocuments: function() {
      return MyDocuments.find({
        author: "Patrick Coffey"
      });
    }
  });
}

// In your template file.
{{#each myDocuments}}
  Title: {{title}},
  Author: {{author}},
  Body: {{body}}
{{/each}}
```

Reactive template system

When documents are inserted, updated, or removed, templating system reacts as expected.

Example: delete a doc, and it no longer appears on interface)

Reactive template system

Optimistic interface

When a dataset on the client changes, the client updates templates to reflect changes.

Doesn't wait for server to respond with "OK"

Package system

atmospherejs.com - Lots of great packages.

```
meteor add userName:packageName
meteor remove userName:packageName
```

Packages are structurally like mini-meteor apps. Very easy to make, install, and maintain.

Build system

Easy to make and transport. (.gz files)

```
meteor build
```

Can be made for different environments…
(cordova)

Meteor and Drupal

There's a lot of talk about Headless Drupal these days...

What about *reactive* headless Drupal .. 8?

Meteor and Drupal

Meteor is well suited to consume data from a Drupal system.

Meteor and Drupal

Drupal is well suited to sync/push data into Meteor's datastore.

Meteor and Drupal

Combining Meteor's reactive interface abilities, and Drupal's highly-organized content model is an excellent way to create reactive interfaces for Drupal websites.

Meteor and Drupal

Let's look at a Drupal API + Meteor front end example application.

Meteor and Drupal

Drupal:

- Mongosync: pushes data to mongodb
- Distill: makes entity data extraction easy
- MongosyncMeteor: implements hooks from Mongosync and uses Distill to turn entity into mongodb-friendly data.

Meteor and Drupal

Meteor:

- MeteorIntroduction: Simple meteor that demonstrates basic capabilities of components discussed today.

Meteor and Drupal

Resources:

- github.com/patrickocoffeyo/mongosync
- github.com/patrickocoffeyo/distill
- github.com/patrickocoffeyo/mongosync_meteor_example
- github.com/patrickocoffeyo/meteor-introduction