

Sviluppo



Di Michele Costabile

*Sviluppo rapido web,
ma anche Android
e iOS, in ambiente
multiplatforma
JavaScript*

Atterriamo su Meteor

Il titolo lo abbiamo scelto per celebrare un grande successo scientifico europeo, l'atterraggio su un sasso di quattro chilometri di diametro a 500 milioni di chilometri di distanza da parte di una sonda spaziale europea. Anche l'oggetto di cui parliamo in questo numero è un successo tecnico interessante, nel suo genere. Meteor è il nome di un framework JavaScript, basato su Node.js e Apache Cordova, che copre le esigenze di sviluppo dal web al mobile con un'unica base di codice. Questa promessa, da sola è un sogno per chi lavora a un sito, già sapendo che gli verrà commissionata anche la app. Ma c'è altro.

Per esempio le applicazioni Meteor sono real time, cioè i client connessi a un server si aggiornano quando cambiano i dati da visualizzare senza ricaricare la pagina, con quell'interazione in tempo reale a cui siamo abituati, per fare un esempio, in Facebook.

In buona sostanza, Meteor è un ambiente comprensivo, unitario, facile e potente da usare, esattamente ciò di cui hanno bisogno le startup, o comunque chi non ha progetti di sviluppo con scadenze molto in là nel tempo.

Dietro il progetto c'è una startup (che ha raccolto la bellezza di 11,2 milioni di dollari di fondi da Y Combinator, una divisione di Yahoo che segue le

aziende esordienti), il *Meteor Development Group* con il progetto, dal quale ci si aspetta un ritorno grazie a un ambiente di hosting dedicato e integrato. Fra i finanziatori figurano Andresseen Horoviz, una società di business angel fondata da Marc Andresseen, noto sul web come un architetto di Netscape, uno dei primi browser sul mercato.

Il progetto Meteor è partito nel 2011 ed è cresciuto rapidamente, acquistando reputazione fra le startup. Oggi, il repository Github di Meteor è seguito da più di 22.000 sviluppatori, le domande con il tag `[meteor]` su StackOverflow.com sono più di 9.200, mentre la home page meteor.com proclama 240.000 installazioni.

I numeri sono interessanti, ma ancor di più lo è il profilo del team di sviluppo (meteor.com/people): una rassegna di rock star della programmazione in startup. Insomma, ci sono i dati di business per dire che il progetto è interessante, ora vediamo quelli tecnici.

I PRINCIPI DI METEOR

I fondamenti che hanno dato forma alla progettazione di Meteor sono sette. *Inviare dati, non pagine.* Meteor manda al client solo i dati necessari per costruire la pagina, ma la costruzione effettiva della vista avviene dal lato del browser.

Esiste comunque uno strato di rendering dal lato server che si può usare per non confondere gli spider dei motori di ricerca.

Un unico linguaggio: JavaScript. Come abbiamo osservato quando abbiamo parlato di Node.js, il JavaScript lato server ha senso e può garantire prestazioni del massimo livello. Consideriamo anche che il linguaggio è popolare, quindi è facile trovare talenti e componenti software. In effetti npmjs.org, il sito che ospita i componenti aggiuntivi per Node.js, dichiara quasi 125.000 pacchetti. L'abbondanza di ruote già inventate porta a un altro dei principi di Meteor: *abbracciare quello che è già stato creato* senza creare inutili alternative a framework e strumenti esistenti, ma invece integrandoli, come il framework jQuery, il linguaggio CoffeeScript, il motore di template Jade, il linguaggio Markdown. *Database ovunque.* L'uso di MongoDB e l'architettura del framework consentono di usare nello stesso modo il database, sia dal lato server, sia dal lato client. Una versione più magra del database, Minimongo, permette di trattare i dati in una cache lato client e usare le stesse primitive che si usano lato server. Su Quora abbiamo trovato un piccolo articolo dell'autore che racconta qualcosa di più sull'architettura.

Compensazione della latenza. Meteor, secondo la documentazione, cerca di anticipare le interrogazioni e le modifiche al database, salvo compiere un rollback, in modo da ridurre il tempo di esecuzione delle query e dare l'impressione che si completino istantaneamente.

Reattività di tutto lo stack. Ogni strato dell'applicazione si aggiorna quando è necessario. Per esempio, quando modifichiamo i sorgenti di un'applicazione in esecuzione, il browser contiene i risultati aggiornati subito dopo la modifica dei sorgenti, mostrando che server e client hanno reagito alla modifica rigenerando il contenuto e ricaricandolo.

Semplicità uguale produttività. Questo principio è il più facile da capire e da confrontare con la propria esperienza. Per esempio, consideriamo l'applicazione `$$skeleton`, una semplice pagina con un pulsante e un testo che indica quante volte il pulsante è stato premuto,

```

<head>
<title>meteor_app</title>
</head>

<body>
<h1>Benvenuto in Meteor!</h1>

{{> hello}}
</body>

<template name="hello">
<button>Clicca qui</button>
<p>Hai premuto il pulsante {{counter}} volte.</p>
</template>

--uu-!----F1 meteor_app.html All L6 (HTML)-----
if (Meteor.isClient) {
  // counter starts at 0
  Session.setDefault('counter', 0);

  Template.hello.helpers({
    counter: function () {
      return Session.get('counter');
    }
  });

  Template.hello.events({
    'click button': function () {
      // increment the counter when button is clicked
      Session.set('counter', Session.get('counter') + 1);
    }
  });
}

if (Meteor.isServer) {
--uu-!----F1 meteor_app.js Top L1 (Java/L Abbrev)-----
Wrote /Users/michele/Src/meteor_app/meteor_app.html

```

Il codice dell'applicazione vuota aperto direttamente nello storico e collaudato editor emacs in una finestra di comando.

```

<template name="appBody">
<div id="container" class="{{menuOpen}} {{cordova}}">

  <section id="menu">
    {{#if currentUser}}
    <div class="btns-group-vertical">
      <a href="#" class="js-user-menu btn-secondary">
        {{#if userMenuOpen}}
        <span class="icon-arrow-up"></span>
        {{else}}
        <span class="icon-arrow-down"></span>
        {{/if}}
        {{emailLocalPart}}
      </a>
      {{#if userMenuOpen}}
      <a class="js-logout btn-secondary">Logout</a>
      {{/if}}
    </div>
  </div>

--uu-!----F1 app-body.html Top L1 (HTML)-----
var MENU_KEY = 'menuOpen';
Session.setDefault(MENU_KEY, false);

var USER_MENU_KEY = 'userMenuOpen';
Session.setDefault(USER_MENU_KEY, false);

var SHOW_CONNECTION_ISSUE_KEY = 'showConnectionIssue';
Session.setDefault(SHOW_CONNECTION_ISSUE_KEY, false);

var CONNECTION_ISSUE_TIMEOUT = 5000;

Meteor.startup(function () {
  // set up a swipe left / right handler
  $(document.body).touchwipe({
    wipeLeft: function () {
      Session.set(MENU_KEY, false);
    },
    wipeRight: function () {
      Session.set(MENU_KEY, true);
    }
  });
});
--uu-!----F1 app-body.js Top L1 (Java/L Abbrev)-----

```

Il codice dell'applicazione *todos* aperto in emacs. Una versione più amichevole si trova su emacsformacosx.com

mostrando il modo in cui si tiene traccia di variabili di sessione con il browser.

Ecco l'esempio di codice

```

<head>
<title>meteor_app</title>
</head>

<body>
<h1>Benvenuto in Meteor!</h1>

{{> hello}}
</body>

<template name="hello">
<button>Clicca qui</button>
<p>Hai premuto il pulsante
{{counter}} volte.</p>
</template>

```

Questo esempio mostra due caratteristiche interessanti di Meteor. La prima è l'inserimento di un template fra le doppie graffe seguite da un segno di maggiore `{{> hello}}`. Il template viene definito di seguito, in un tag stile html. All'interno del codice del template una variabile di sessione, chiamata *counter*, viene incorporata nel codice, come potremmo fare scrivendo `$counter` in Php. Quello che non c'è nell'esempio è significativo: non abbiamo indicazioni del foglio di stile associato alla pagina, non includiamo file JavaScript, non c'è un tag `<html>`.

Queste cose non ci sono perché non è necessario metterle: ci pensa il framework e il foglio di stile associato alla pagina è gestito correttamente dal browser. Per concludere, nella pagina

riservata al progetto su meteor.com, l'obiettivo del gruppo è espresso così: «Creare una piattaforma per lo sviluppo di applicazioni cloud che diventi ubiqua, come le piattaforme Unix, http e il database relazionale». Si tratta di un obiettivo molto ambizioso, ma la missione di una startup deve essere ambiziosa, così come lo era *un computer su ogni scrivania* per Microsoft. Inoltre, in Meteor hanno i fondi e le persone con cui possono mirare a raggiungere l'obiettivo.

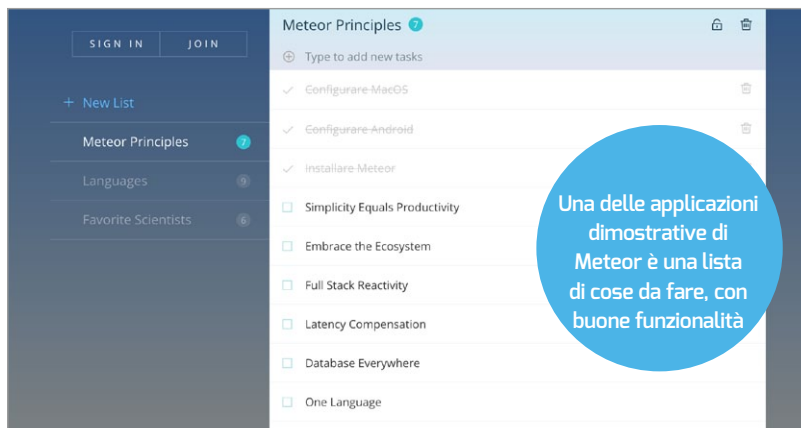
COME SI LAVORA CON METEOR

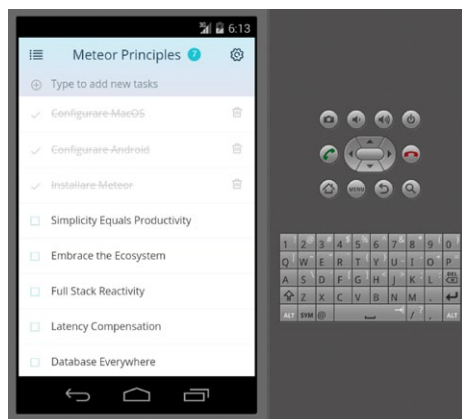
Per usare Meteor serve un sistema Unix o un Mac. Un ambiente per Windows è in arrivo, ma al momento è ancora in beta. Ci si può registrare per avere una notifica quando sarà rilasciato, oppure si può iniziare a sperimentare con il file puntato dalla pagina github.com/meteor/meteor/wiki/Preview-of-Meteor-on-Windows.

L'installazione, per il momento, è quanto di più Unix si possa immaginare: una url che punta a uno shell script da lanciare da riga di comando, così:

```
curl https://install.meteor.com | /bin/sh
```

Lo script, 216 righe in totale, provvede





L'applicazione dimostrativa in esecuzione nell'emulatore Android

a scaricare e installare tutto il necessario verificando che il sistema operativo sia OS X o Linux e che non ci siano versioni troppo vecchie del pacchetto già installate. Dopo l'installazione siamo pronti per creare uno scheletro di applicazione, con

```
meteor create meteor_app
```

Questo crea una directory chiamata meteor_app nella posizione in cui siamo, per esempio \$HOME/src, e crea i file di base per un'applicazione web, cioè

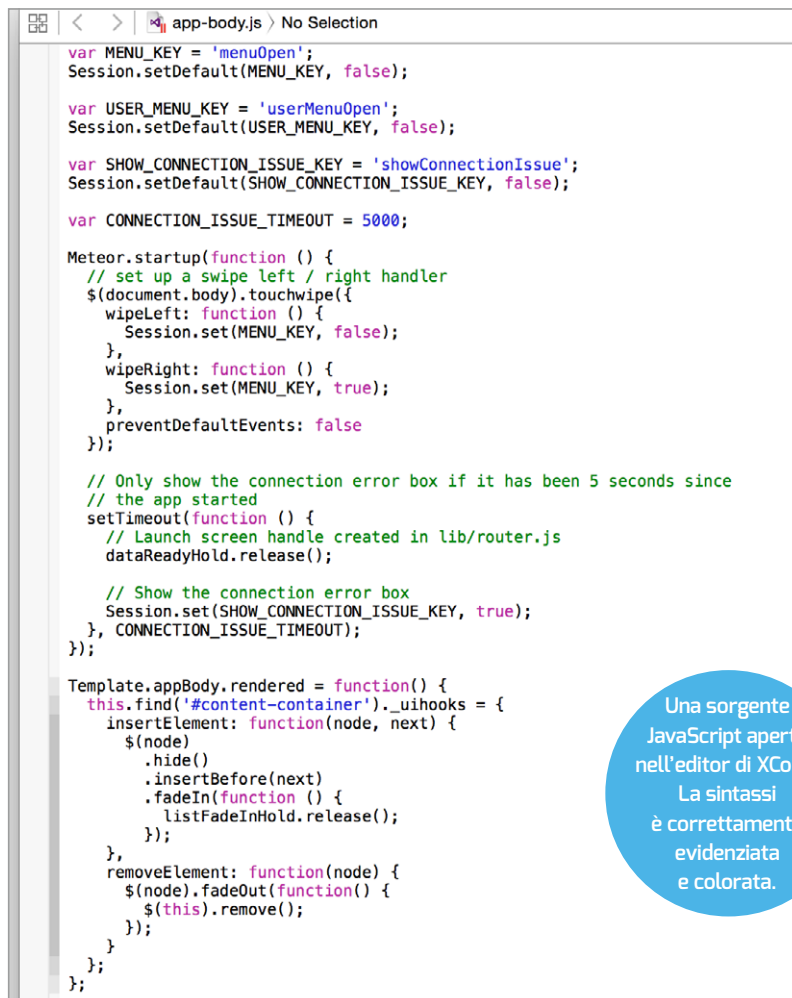
```
meteor_app.css
meteor_app.html
meteor_app.js
```

Per vedere l'applicazione in esecuzione, basta eseguire il comando *meteor* e aprire il browser all'indirizzo localhost:3000. Il file html è quello che abbiamo mostrato prima e contiene uno scheletro di pagina e un semplice template. Il codice JavaScript è altrettanto breve:

```
if (Meteor.isClient) {
  // imposta il contatore a 0
  Session.setDefault('counter', 0);

  Template.hello.helpers({
    counter: function () {
      return Session.get('counter');
    }
  });

  Template.hello.events({
    'click button': function () {
      // incrementa il contatore
    }
  });
}
```



```
quando si preme il pulsante
  Session.set('counter',
    Session.get('counter') + 1);
  });
}
```

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    // codice che verrà eseguito sul server
  });
}
```

Notiamo due cose interessanti. La prima è che il codice è diviso in due sezioni, una per il client e una per il server. L'applicazione, quindi, è unitaria a un livello profondo. La seconda è che il codice client, oltre a un'inizializzazione, contiene funzioni destinate a innestarsi su un template di nome hello, perché entrambe iniziano con Template.hello. La parte helpers definisce variabili per il template, in questo caso una sola, chiamata counter. L'argomento fra le parentesi tonde di Template.hello.helpers è la sintassi JavaScript per definire un oggetto anonimo e inizializzare una proprietà chiamata counter. Si tratta di uno di quei casi in cui ci ripetiamo che JavaScript è un Lisp con le graffe, che

è un complimento. Questa proprietà verrà usata nella parte del template fra doppie graffe, come mostrato nelle linee di codice che ripetiamo qua sotto:

```
<template name="hello">
  <button>Clicca qui</button>
  <p>Hai premuto il pulsante
  {{counter}} volte.</p>
</template>
```

A questo punto, dovrebbe essere chiaro come si usa la sostituzione di variabili tipica di ogni linguaggio web, da Asp e Php in poi. Per quello che riguarda gli eventi, l'agancio è automatico. Come nell'esempio, si crea un oggetto con delle proprietà che sono selettori. In questo caso

```
Template.hello.events({
  'click button': function () {
    // incrementa il contatore
    quando si preme il pulsante
    Session.set('counter',
      Session.get('counter') + 1);
  }
});
```

il selettore è 'click button'. La prima parte è il nome dell'evento, che può

essere appunto click, focus, keypress, eccetera. Dopo lo spazio c'è un selettore, in questo caso *button*, per una classe di oggetti a cui attaccare l'evento. Potremmo essere più specifici e dare una classe al pulsante

```
<button class="my-button">My
button</button>
```

e rendere più preciso il selettore dell'evento

```
"click .my-button": function
(event, template) {
  alert("My button was
clicked!");
}
```

Questa sintassi dei selettori permette di unificare il codice, mettendo tutte le uova nello stesso cesto, all'interno di una funzione `top Template.name.events`. Nello stesso tempo, c'è la flessibilità che occorre per mettere ogni funzione al suo posto. Il template e il suo codice associato sono facilmente trattabili come unità di codice.

ANDROID E IOS

Dopo aver dato una veloce idea di come funziona il lato html di Meteor, è ora di tornare sull'aspetto più interessante, la possibilità di non limitarsi a gestire il codice di client e server, ma anche aggiungere due piattaforme mobili, di gran lunga le più diffuse, grazie all'integrazione di Apache Cordova.

Lo sviluppo di applicazioni mobili, nella documentazione è descritto sostanzialmente solo per unix: sia Linux, sia OS X. La creazione di applicazioni per iOS richiede espressamente un Mac, perché la catena di compilazione per iOS è legata a XCode e OS X, mentre lo sdk di Android può essere installato ovunque.

Il primo passo consiste nell'installazione degli ambienti di sviluppo

```
meteor install-sdk android #
per Android
meteor install-sdk ios #
per iOS
```

Nel caso di iOS dobbiamo completare l'installazione lanciando XCode e accettando le condizioni di licenza, mentre lo sdk per Android non ha bisogno di rifiniture.

Il secondo passo nell'aggiunta del supporto per le piattaforme, cioè dei package meteor necessari per lo sviluppo mobile.

Ricordiamoci che Meteor eredita da Node.js un eccellente sistema di gestione dei package, chiamato npm.

```
meteor add-platform android #
per Android
meteor add-platform ios #
per iOS
```

Adesso, possiamo mandare in esecuzione l'applicazione di test sugli emulatori delle due piattaforme

```
meteor run android #
per Android
meteor run ios #
per iOS
```

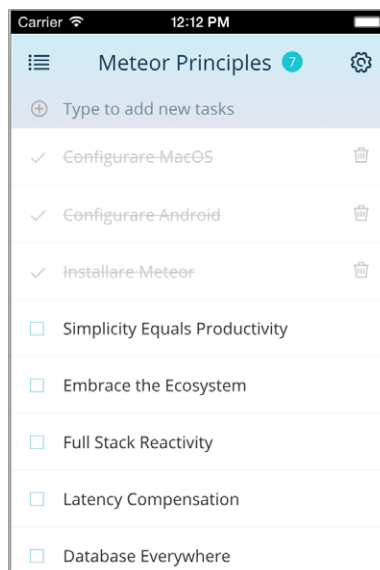
Meteor ci assiste anche nel lancio dell'applicazione su veri device connessi al sistema

```
meteor run android-device #
per Android
meteor run ios-device #
per iOS
```

IL VERDETTO

Meteor è una grossa novità ed è opportuno tenerla nel radar strategico. Sul web, Meteor è stato definito *quello che è stato Rails per Ruby*, una definizione che coglie alcuni punti di contatto, precisamente l'estensione di un concetto elegante a un'eleganza diffusa in tutta la catena di sviluppo.

Usare un solo linguaggio per client



La stessa applicazione eseguita nell'emulatore Apple di iOS

e server ha dei vantaggi indiscutibili. In questa epoca che tende a favorire i client robusti, nei team sono richieste conoscenze di qualità sia dal lato client, sia dal lato server. Se il linguaggio e le tecniche sono le stesse, ci sono benefici indiscutibili.

I PUNTI DI FORZA SONO SOLIDISSIMI

Il framework non è una promessa. Nonostante sia sempre in crescita è già maturo e le tecnologie su cui si appoggia, come Node.js e Cordova, sono quelle che comunque sono rilevanti per qualsiasi nuovo progetto. Il team di sviluppo è notevole e la comunità è molto attiva, questo significa che non solo c'è abbondanza di documentazione, ma esistono anche numerosi siti satelliti, come crater.io, eventedmind.com, meteortips.com. In particolare, segnaliamo una introduzione ai punti di forza dell'ambiente alla url bit.ly/whymeteor.

La base che offre Node.js è notevole, per esempio possiamo procurarci il necessario per gestire gli account sul nostro sito, con la scelta fra account nativi e account offerti dalle principali piattaforme che supportano openID, con questi comandi

```
meteor add accounts-password
meteor add accounts-twitter
meteor add accounts-google
meteor add accounts-facebook
```

e infine aggiungere una infrastruttura di interfaccia utente per il login con

```
meteor add accounts-ui
```

Questo è molto meglio che rivolgere uno sguardo a metà fra condiscendente e disperato al cliente e al direttore tecnico che chiedono il supporto per un login più semplice.

Abbiamo anche già visto quanti dettagli, come una accurata compilazione degli header, sono resi non necessari dagli automatismi del framework. La semplicità con cui si crea uno scheletro di applicazione e quella con cui si riveste lo scheletro di bellezza, con un po' di css, sono un altro elemento importante: è facile e rapido avere qualcosa da far vedere, una funzione molto importante con i tempi che corrono e fanno correre.

Infine aggiungiamo a multipiattaforma un'altra parola chiave su cui riflettere: real time.