

Getting Started with node.js

Beardsley (“B”) Ruml

b@ruml.com

<http://ruml.com>

probably the most
exciting new idea
since Ruby on Rails
in 2004

node.js

2009

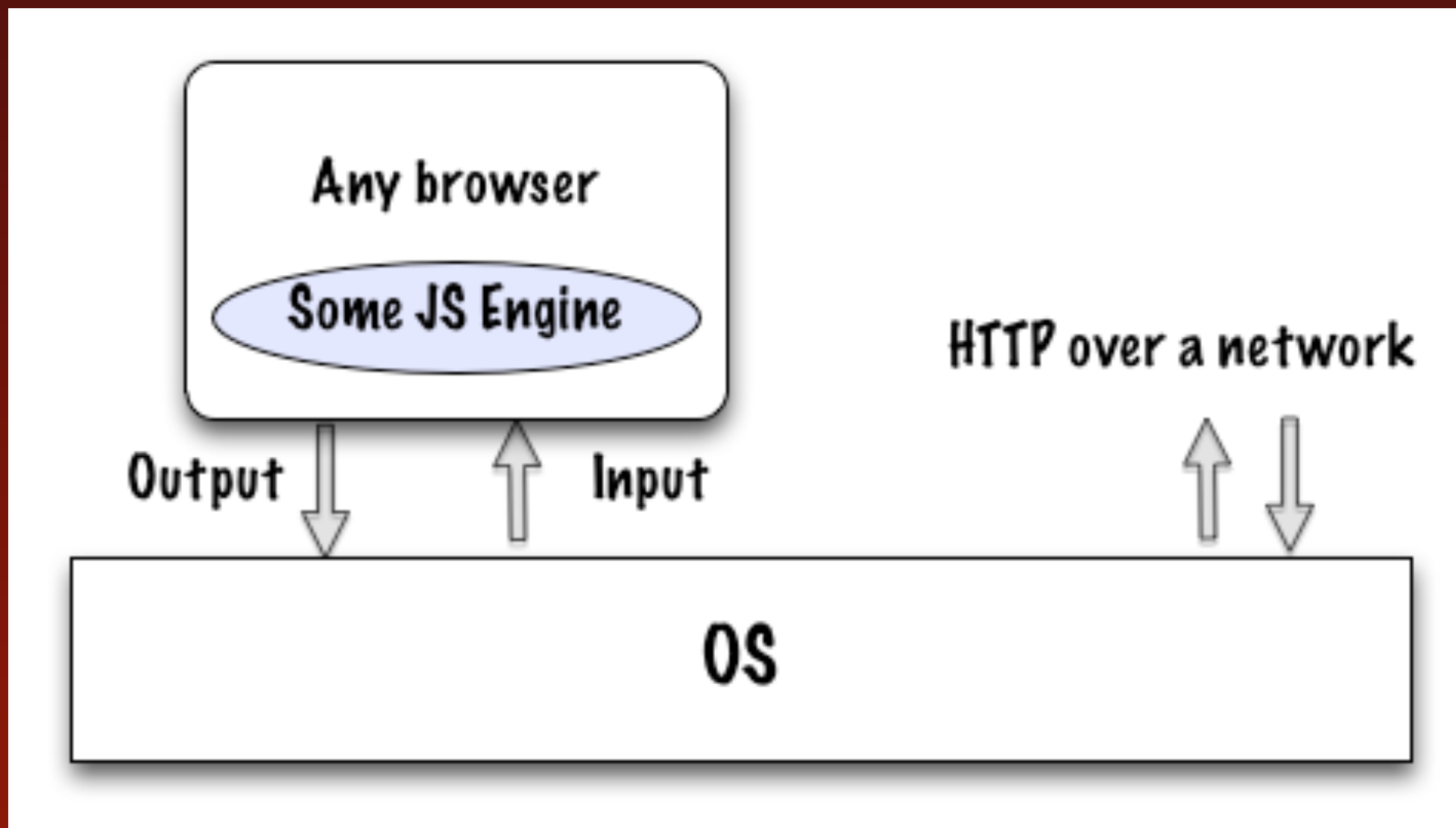
Ryan Dahl

What is node.js?

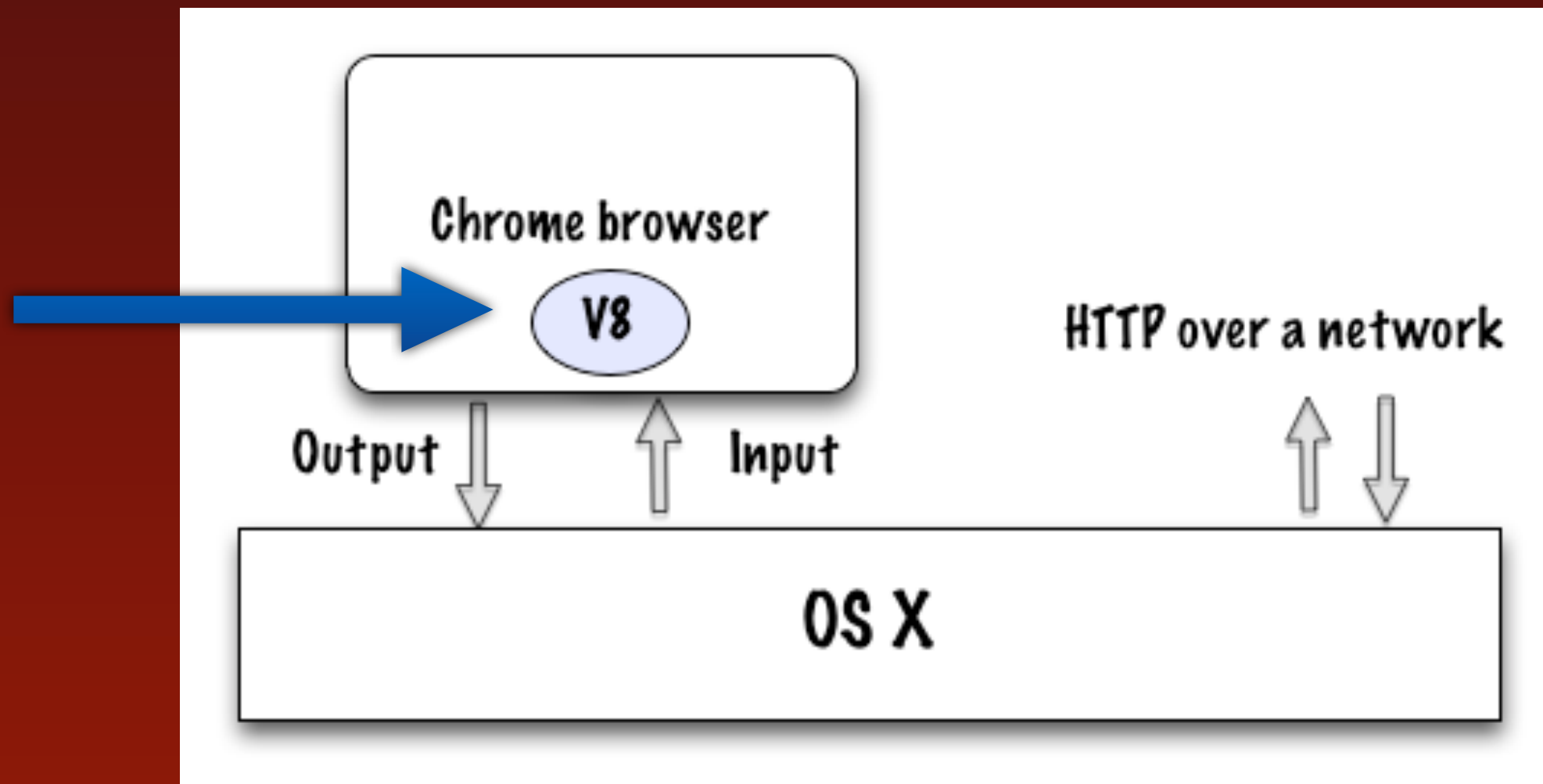
What is node.js?

a program execution environment for JS

A browser is a program execution environment for JS

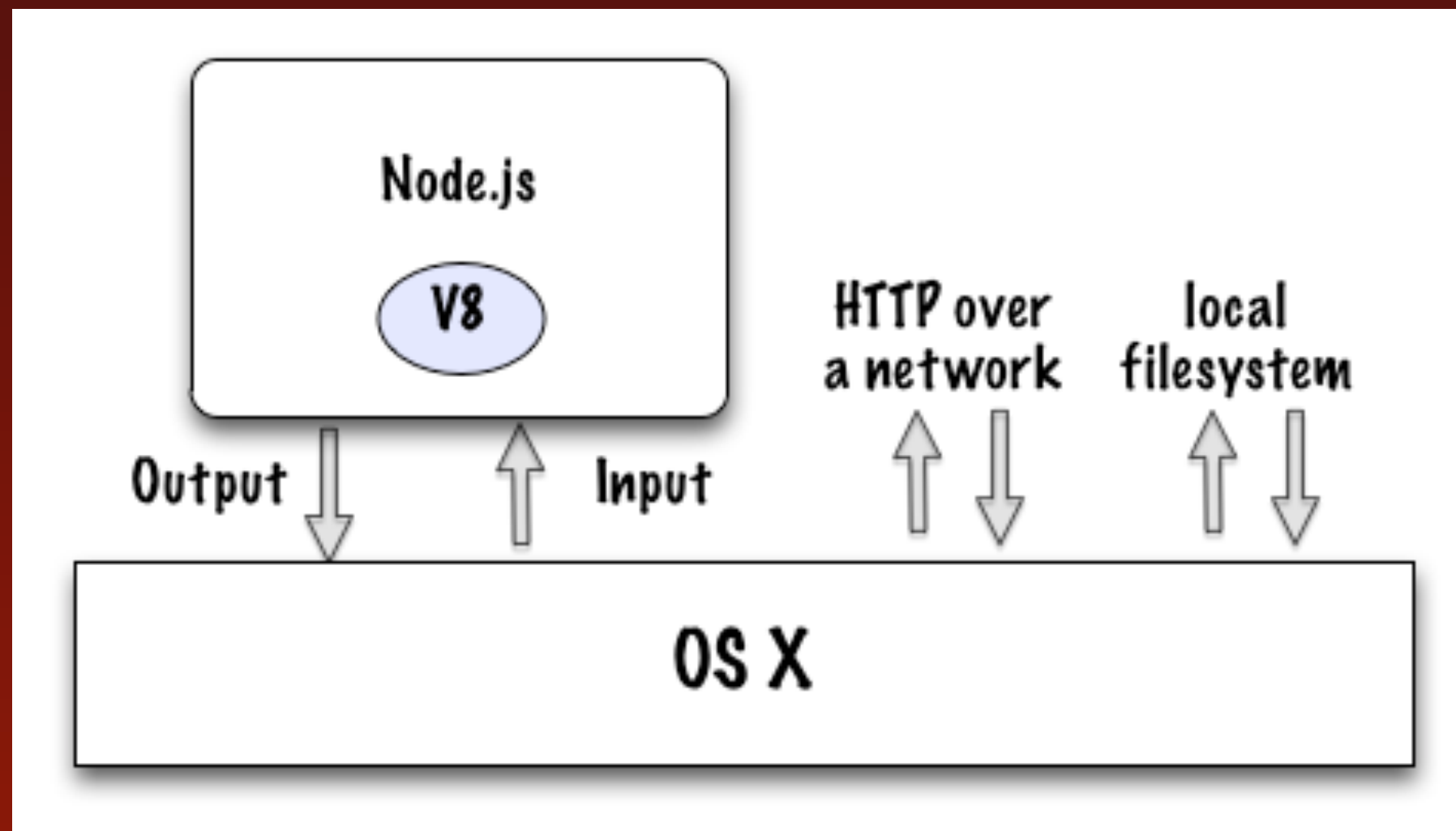


The Chrome browser is a program execution environment for JS



node.js is

a program execution environment for JS



with access to the local filesystem

What is node.js?

a program execution environment for JS
with event-driven, non-blocking I/O

event-driven:

On launch, the program sits in an event-loop waiting for GUI events to occur; when they do, each event is placed at the end of an event-queue.

event-driven(2):

The program removes each event from the head of the queue and “handles” it by invoking the event-handler functions which have been bound to the event.

“event-handler”

==

“callback”

The program “calls back”
when the event occurs.

non-blocking I/O:

When input is requested (e.g., disk-access), the program doesn't wait for the data from disk. It continues on with "something else."

non-blocking I/O:

The code to be executed when the disk data is finally available is placed in a “callback function” and that function is invoked when the data-available event occurs.

What is node.js?

a program execution environment for JS
with event-driven, non-blocking I/O
written in C++ and incorporating
Google's V8 JavaScript engine
[also a set of core modules]

Why is it so fast as a network server?

In CGI, each request spawns a new thread with a separate instance of the application: new interpreter, new initialization, etc.

In node.js, each request triggers a callback within a single thread (small heap memory allocation) which provides an environment which can save state.

Why is node.js “the future” of web applications?

Web evolution:

- a series of linked static pages;
- pages which were dynamically generated from a database (templating);
- pages which send significant amounts of new data and can ask for updates for parts of a page without refreshing the page (Ajax);
- pages which need constant communication; updates without asking; many browsers talking to each other (chat, backchannel);

node.js core modules

http

net

child_process

fs

os

sys

url

util

...

require(<module>)

```
var http = require("http");
```

[convention: use the same name as module]

a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
});
server.listen(1337);
console.log('Server running on port 1337');
```

a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
});
server.listen(1337);
console.log('Server running on port 1337');
```

```
[rum1@TheBoss-784 code]$ node helloWorldServer.js
Server running on port 1337
```

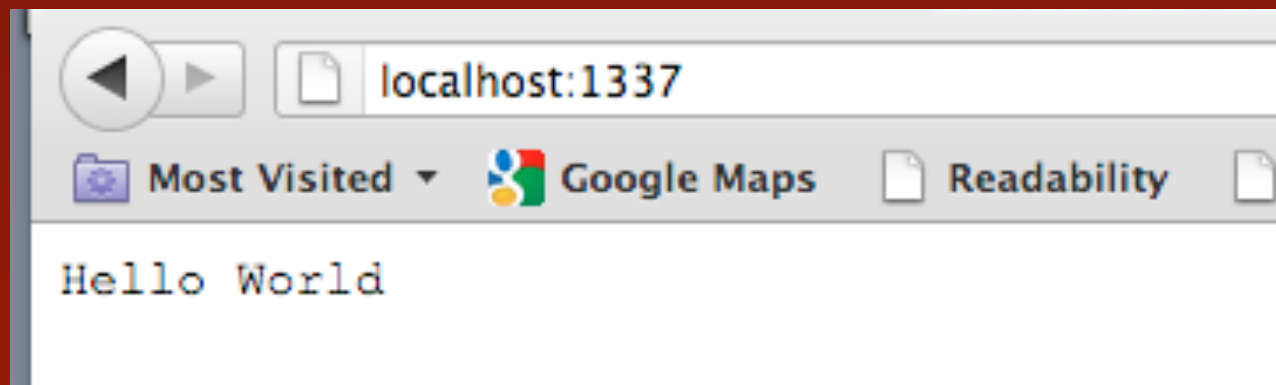


a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
});
server.listen(1337);
console.log('Server running on port 1337');
```

```
[ruml@TheBoss-784 code]$ node helloWorldServer.js
Server running on port 1337
```



what does the call to createServer() do?

```
server = http.createServer(function (request, response) {  
  response.writeHead(200, { 'Content-Type': 'text/plain' });  
  response.end('Hello World\n');  
});
```


what does the call to createServer() do?

```
server = http.createServer(function (request, response) {  
  response.writeHead(200, { 'Content-Type': 'text/plain' });  
  response.end('Hello World\n');  
});
```

It binds the event-handler function
(passed as the parameter) with the
incoming-HTTP-request event.

a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
});
server.listen(1337);
console.log('Server running on port 1337');
```

but we can chain our calls!

a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(1337);
console.log('Server running on port 1337');
```

a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
}).listen(1337);
console.log('Server running on port 1337');
```

what's wrong here?

a web server in node:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
}).listen(1337, function() {
  console.log('Server running on port 1337');
});
```

the serverResponse object:

```
var http = require('http');

server = http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
}).listen(1337, function() {
  console.log('Server running on port 1337');
});
```

the serverRequest object:

- [request.method](#)
- [request.url](#)
- [request.headers](#)
- [request.trailers](#)
- [request.httpVersion](#)
- [request.setEncoding\(encoding=null\)](#)
- [request.pause\(\)](#)
- [request.resume\(\)](#)
- [request.connection](#)

let's log the requests:

```
var http = require('http');

function logRequest(request) {
  console.log("REQUEST: " + request.method + " HTTP " +
    request.httpVersion + " " + request.url);
  console.dir(request.headers);
}

server = http.createServer(function (request, response) {
  logRequest(request);
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(1337, function() {
  console.log('Server running on port 1337');
});
```

let's log the requests:

```
^C[rum1@TheBoss-784 code]$ node helloWorldServer.js
Server running on port 1337
REQUEST: GET HTTP 1.1 /
{ host: 'localhost:1337',
  'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X
accept: 'text/html,application/xhtml+xml,application/
'accept-language': 'en-us,en;q=0.5',
'accept-encoding': 'gzip, deflate',
'accept-charset': 'ISO-8859-1,utf-8;q=0.7,*;q=0.7',
connection: 'keep-alive',
'cache-control': 'max-age=0' }
```


let's serve a static file:

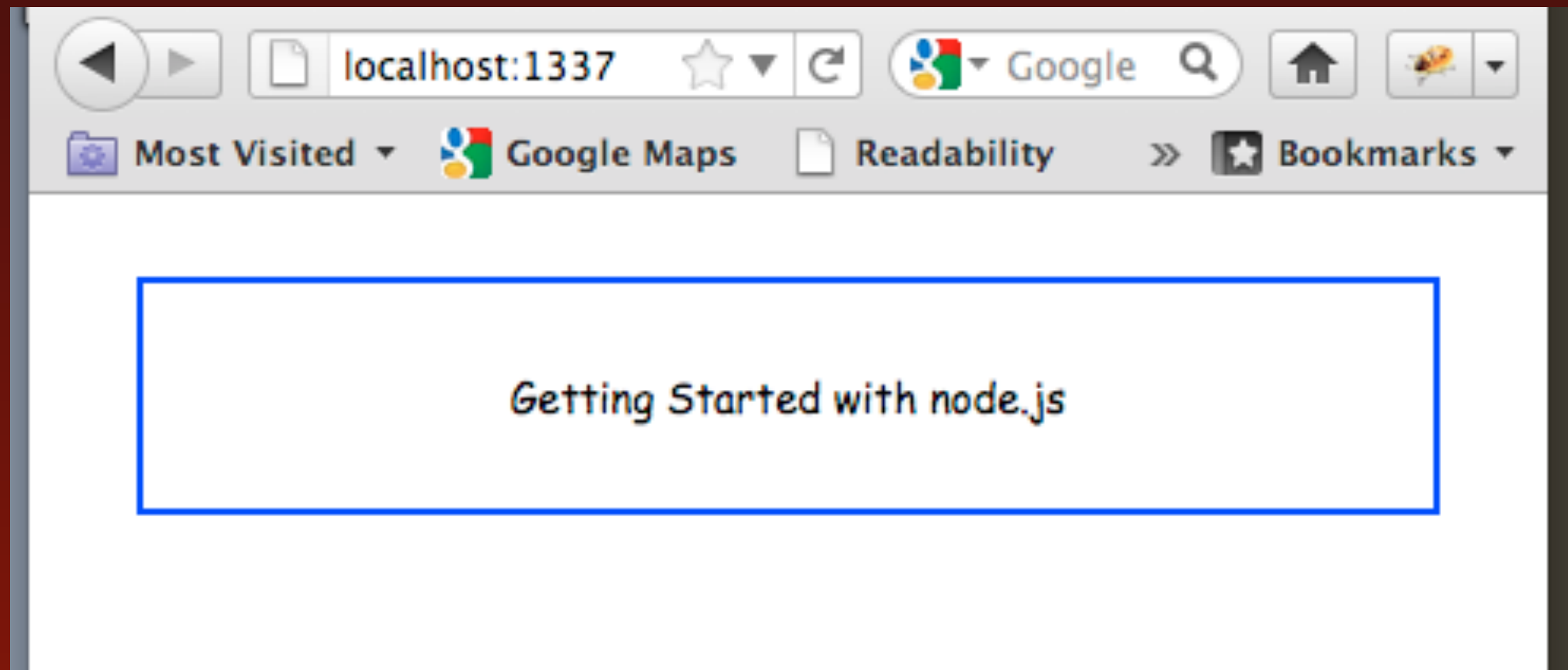
```
var http = require('http');
var fs   = require('fs');

server = http.createServer(function (request, response) {
  fs.readFile('./home.html', function(err, data) {
    response.writeHead(200, { 'Content-Type': 'text/html' });
    response.end(data);
  });
}).listen(1337, function() {
  console.log('Server running on port 1337');
});
```

let's serve a static file:

```
<html>
<head>
  <title>Getting Started with node.js</title>
</head>
<body>
  <div id="container"
    style="margin: 2em;
          border: 2px solid blue;
          padding: 2em;
          text-align: center;">
    Getting Started with node.js
  </div>
</body>
</html>
```

let's serve a static file:



Web App Framework: Express

```
var app = require("express").createServer();

app.get('/', function(req, res){
  res.send('Hello World');
});

app.listen(3000);
```

A Chat Application

(the “demo app” of node.js)

How does chat work?

Joe, Sue and Bob all go to the same URL.
They navigate to the same “chat room.”

How does chat work?

How does chat work?

1. When anyone sends a message, the sender's ID and the message are displayed in the browsers of all. (In the sender's browser, the sender is "you.")

How does chat work?

1. When anyone sends a message, the sender's ID and the message are displayed in the browsers of all. (In the sender's browser, the sender is "you.")
2. When a new person enters (or leaves) the room, everyone else is notified.

How does chat work?

1. When anyone sends a message, the sender's ID and the message are displayed in the browsers of all. (In the sender's browser, the sender is "you.")
2. When a new person enters (or leaves) the room, everyone else is notified.
3. The newcomer is sent the most recent messages.

This isn't typical client-server!

**This requires a:
permanently-open
bidirectional
channel.**

Before HTML5: hacks!

long polling
flash

After HTML5: websockets!

(a
permanently-open
bidirectional
channel)

websockets for node.js:

socket.io!

5 different transports
(including long polling and flash)

JavaScript object syntax:

```
{  
  firstName: "John",  
  lastName: "Brown",  
  age: 23,  
  children: [ "Sue", "Bob" ],  
  birthdate: [ 1994, 12, 25 ]  
}
```

back to the chat app:

What messages do we need to send?

What messages do we need to send?

1. When anyone sends a message, the sender's ID and the message are displayed in the browsers of all. (In the sender's browser, the sender is "you.")

Message from the sender's browser:

`<body>`

Message from the server to others:

```
{ message: [<author>, <body>] }
```

What messages do we need to send?

2. When a new person enters (or leaves) the room, everyone else is notified.

Message from the server to others:

```
{ announcement: <text> }
```

What messages do we need to send?

3. The newcomer is sent the most recent messages.

Message from the server to newcomer:

```
{ buffer: [ <msg>, <msg>, ... ] }
```

Chat app messages summary:

Messages from the browser:

`<body>`

Messages from the server:

```
{ buffer: [ <msg>, <msg>, ... ] }
```

```
{ announcement: <text> }
```

```
{ message: [<author>, <body>] }
```

What has to happen on the server?

1. Listen for attempts to connect.
2. When a connection occurs:
 - create a client object and assign an ID;
 - bind event-handlers to the events of the client: 1) receipt-of-a-message and 2) disconnecton
3. The message event-handler needs to:
 - add the client's ID as author;
 - package author and body into a message object;
 - sent the message object to everyone else.

Let's look at the server code:

```
10 var serverListener = require('socket.io').listen(server);
11 var buffer = [];
12
13 serverListener.on('connection', function(client){
14   // send the buffer to this client;
15   client.send({ buffer: buffer });
16   // send an announcement to all other clients;
17   client.broadcast({ announcement: '=> ' + client.sessionId + ' connected' });
18
19   client.on('message', function(message){
20     var msg = { message: [client.sessionId, message] };
21     buffer.push(msg);
22     // maximum of 15 messages in the buffer at once;
23     if (buffer.length > 15) buffer.shift();
24     client.broadcast(msg);
25   });
26
27   client.on('disconnect', function(){
28     client.broadcast({ announcement: client.sessionId + ' disconnected' });
29   });
30 });
```

What has to happen on the browser?

1. Create a socket object and send a connection request to the server.
2. Bind event-handler to receipt-of-message event.
3. The message event-handler needs to:
 - for buffer objects: add to chat window in a loop;
 - otherwise just add to chat window.

Let's look at the browser code:

```
4 <script src="/js/socket.io.js"></script>
5 <script>
6   var socket = io.connect('http://localhost:8888');
7   socket.on('message', function(obj){
8     if ('buffer' in obj){
9       for (var i in obj.buffer) { addParaToChat(obj.buffer[i]); }
10    } else {
11      addParaToChat(obj);
12    }
13  });
14
15  socket.on('connect', function(){
16    addParaToChat({ message: ['System', 'Connected']})
17  });
18  socket.on('disconnect', function(){
19    addParaToChat({ message: ['System', 'Disconnected']})
20  });
21 </script>
```


backchannel

A variation on chat: posts (typically questions for a lecturer) are voted on by all and can be displayed in order of popularity.

<http://github.com/brum12/backchannel>

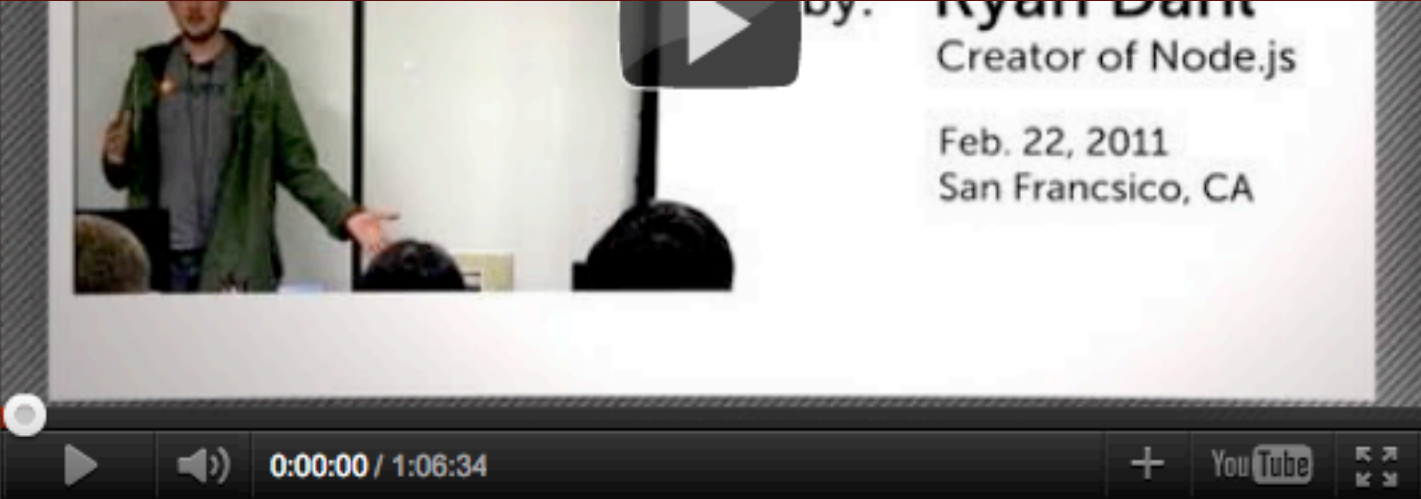
node.js installation

<http://nodejs.org>

[Download](#)
[ChangeLog](#)
[About](#)
[v0.6.2 docs](#)

[Wiki](#)
[Blog](#)
[Community](#)
[Demo](#)
[Logos](#)
[Jobs](#)

- [shutterstock](#)
- [Yahoo Inc](#)
- [frog design inc.](#)
- [Klout](#)
- [Ming.ly](#)



by: **Ryan Dahl**
Creator of Node.js
Feb. 22, 2011
San Francisco, CA

0:00:00 / 1:06:34

Download

2011.11.18 v0.6.2

- [node-v0.6.2.tar.gz](#) **Source code (build instructions)**
- [node-v0.6.2.msi](#) **Windows installer**
- [node-v0.6.2.pkg](#) **Macintosh installer**
- [Documentation](#)
- [Other release files \(like .exe and .pdb\)](#)

Historical: [versions](#), [docs](#)

node.js documentation

<http://nodejs.org/docs/v0.6.2/api/all.html>

Node.js v0.6.2 Manual & Documentation

[Index](#) | [View on single page](#)

Table of Contents

- [Synopsis](#)
- [Globals](#)
- [STDIO](#)
- [Timers](#)
- [Modules](#)
- [C/C++ Addons](#)
- [Process](#)
- [Utilities](#)
- [Events](#)
- [Buffers](#)
- [Streams](#)
- [Crypto](#)
- [TLS/SSL](#)

node.js directory

nodecloud.org



NodeCloud is a resource directory gathering sites related to Node.js and ordering them by their Alexa traffic, allowing to evaluate relative popularity of a project. Screenshots are generated locally using [PhantomJS](#).

To suggest new sites to be added to the listing, you can reach me through my [site](#) or on [twitter](#).

Note to programmers : you might as well be interested in my [Ascii Codes](#) reference chart.

BREAKING NEWS : check out [Echo Linux](#), a social news site dedicated to Linux and related topics!

Share this site :





Node.js
Home of the Node.js project, a network application framework written on top of Google V8 JavaScript engine.
NodeCloud Ranking : 1
Alexa Traffic Rank : 19039

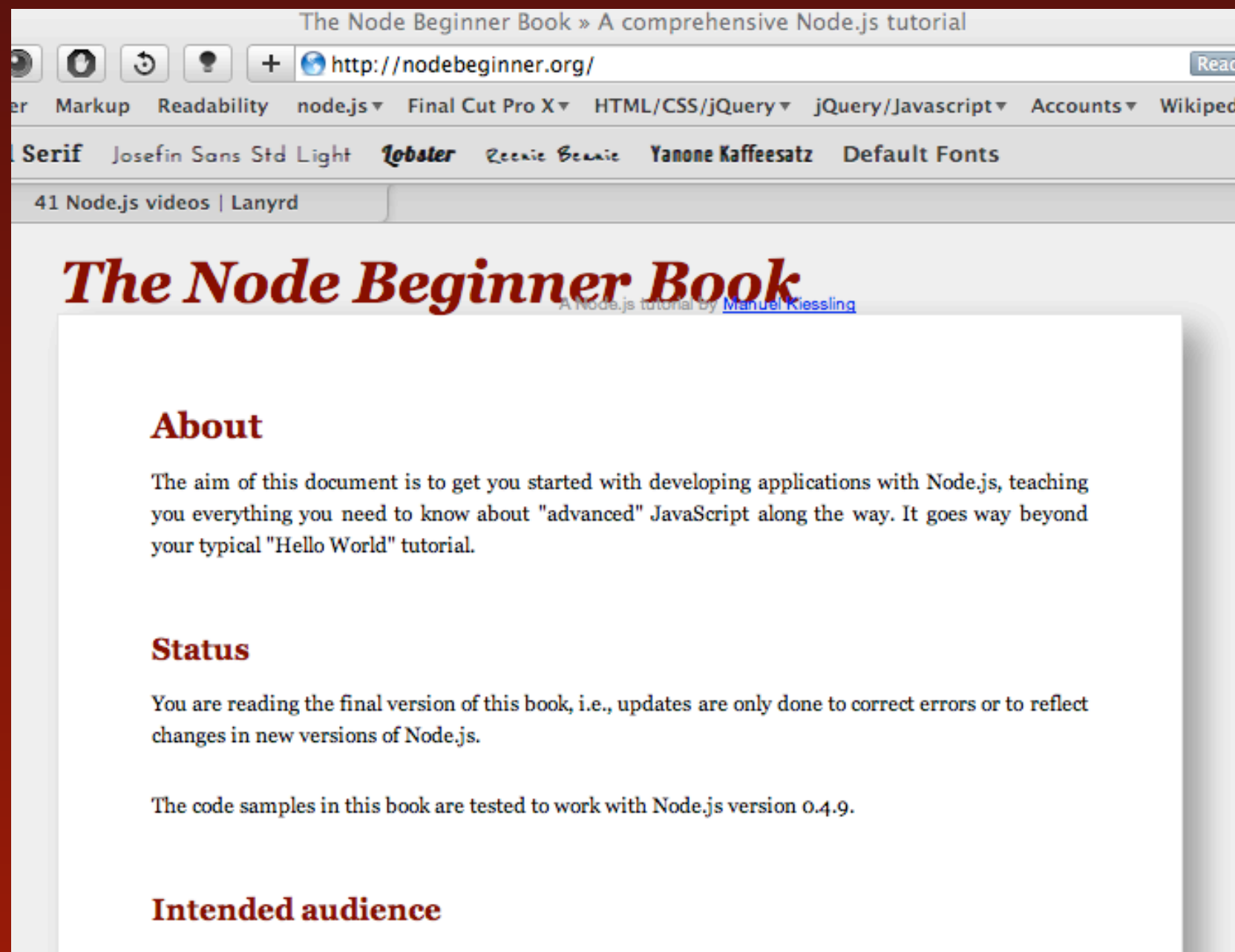


Node SmartMachines
Node.js cloud hosting services by Joyent, offering free Node SmartMachines.

node.js tutorials

The Node Beginner Book

<http://nodebeginner.org>



node.js tutorials

Mastering Node

[http://visionmedia.github.com/masteringnode/
book.html](http://visionmedia.github.com/masteringnode/book.html)

Mastering Node

[Node](#) is an exciting new platform developed by *Ryan Dahl*, allowing JavaScript developers to build high performance servers by leveraging [Google's V8](#) JavaScript engine, and asynchronous I/O. In this book you will discover how to write high concurrency web servers, utilizing the CommonJS module system, third party modules, high level web development and more.

Installing Node

In this chapter we will be looking at the installation and compilation of node. Although there are many ways to install node, we will be looking at [homebrew](#), [nDistro](#), and the most flexible method, of course, source.

Homebrew


Homebrew is a package management system for *OSX* written in Ruby, is extremely well adapted to install node via the `brew` executable simply run:

```
$ brew install node.js
```

node.js tutorials

Node Tuts screencasts

<http://nodetuts.com>



The screenshot shows the Node Tuts website interface. At the top, the title "NODE TUTS" is displayed in large, bold, black letters. To the right of the title are three navigation links: "HOME", "EPISODES", and "CONTACTS". Below the title, the "LATEST EPISODE" section features the title "Deploying to Nodejitsu" by Pedro Teixeira, dated 2011-11-04. A description states: "In this episode Pedro shows how you can deploy your Node app to the Nodejitsu cloud." To the right of this section, it says "Node Tuts is sponsored by:" followed by the "questionform" logo. Below the sponsorship, it says "Also, check out my e-book:" followed by a book cover titled "HANDS-ON NODE.JS". The bottom section lists "EPISODE 28" titled "Javascript classes, prototypes and closures" by Pedro Teixeira, dated 2011-09-26. Below this, it says "In this episode Pedro digs into Javascript prototypical chaining, pseudo-classes and closures." The next episode listed is "EPISODE 27" titled "Kyo Jobs" dated 2011-07-27.

NODE TUTS

[HOME](#) [EPISODES](#) [CONTACTS](#)

LATEST EPISODE 2011-11-04

Deploying to Nodejitsu
by Pedro Teixeira

In this episode Pedro shows how you can deploy your Node app to the Nodejitsu cloud.

Node Tuts is sponsored by:

questionform

Also, check out my e-book:

**HANDS-ON
NODE.JS**

EPISODE 28 2011-09-26

Javascript classes, prototypes and closures
by Pedro Teixeira

In this episode Pedro digs into Javascript prototypical chaining, pseudo-classes and closures.

EPISODE 27 2011-07-27

Kyo Jobs

node.js tutorials

How To Node

<http://howtonode.org>

HowTo Node

The zen of coding in node.JS

A Simple Blog with CouchDB, Bogart, and Node.js


By Nathan Stott  2011.09.19

Update: By request I have posted a [gist of the app.js](#) using *MongoDB* instead of *CouchDB*. This gist also serves as a beginning example for how to use non-promise-based APIs with bogart.

In this article, you will learn how to use Bogart and CouchDB to create a minimal blogging engine. The Express with MongoDB article was a huge hit. This article has similar goals but shows a different way of using Node.JS.

[Read more...](#)


Fun Putting Node on Mobile Devices

By Tim Caswell  2011.07.01

This article will walk you through creating an Ubuntu image that can be chrooted inside a mobile device like the recently released [TouchPad](#). Once the Ubuntu environment is setup we'll learn how to compile and install node for fun and/or profit.

[Read more...](#)

How To Module

By Isaac Z. Schlueter  2011.02.28

These are some basic steps for writing a NodeJS module.

Most of the suggestions in this document are optional. You can definitely write your program however you like, and many in the node community enjoy trying out new creative ways of doing things.

This is merely a set of patterns that noders have found to work for them and their projects.

About HowToNode.org

HowToNode.org is a community supported blog created by [Tim Caswell](#). The purpose of the blog is to teach how to do various tasks in [node.js](#) as well as teach fundamental concepts that are needed to write effective code.

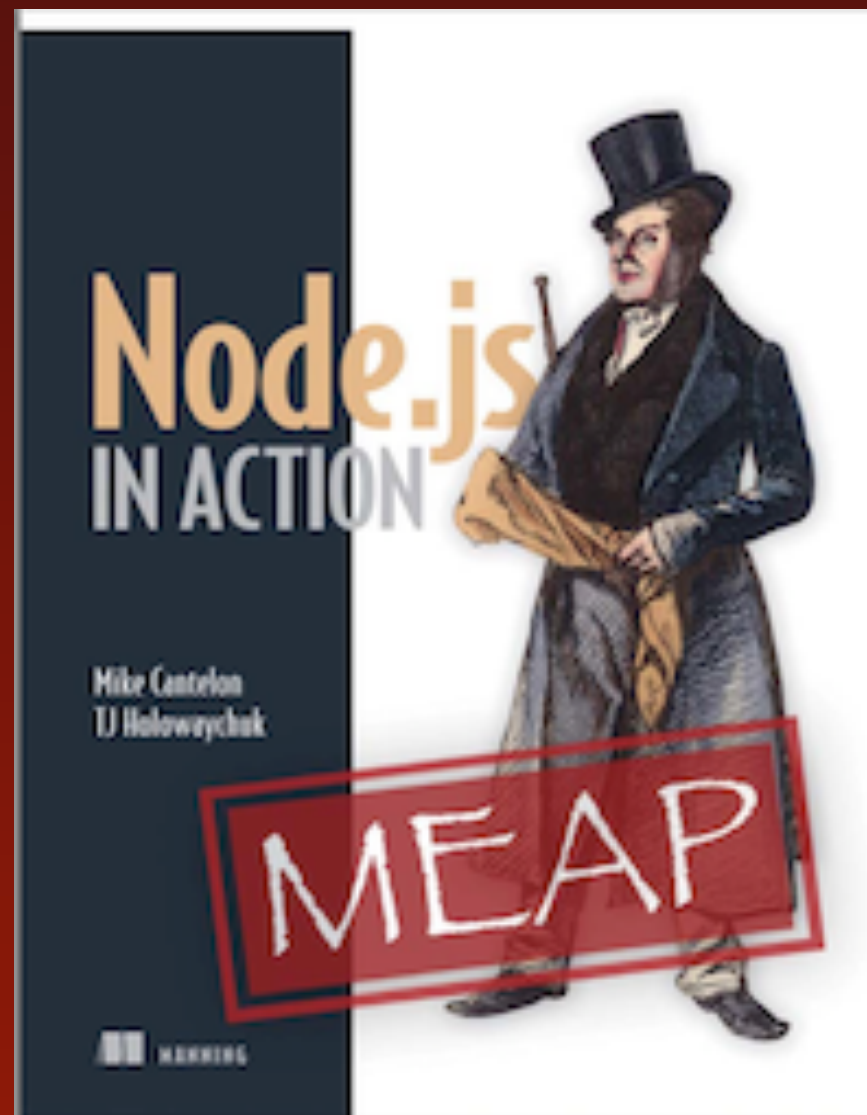
This site is powered by [Wheat](#), a git based blogging engine written in [node.JS](#).

The content for this site is stored in a [git repository](#) that anyone can fork, write an article, and send a pull request. If your article passes the quality standards it will be published and help support the greater node community.

Articles

node.js books

Node.js in Action (Manning)



node.js books

Hands-on node.js

60 pages (of 118) free

<http://nodetuts.com>

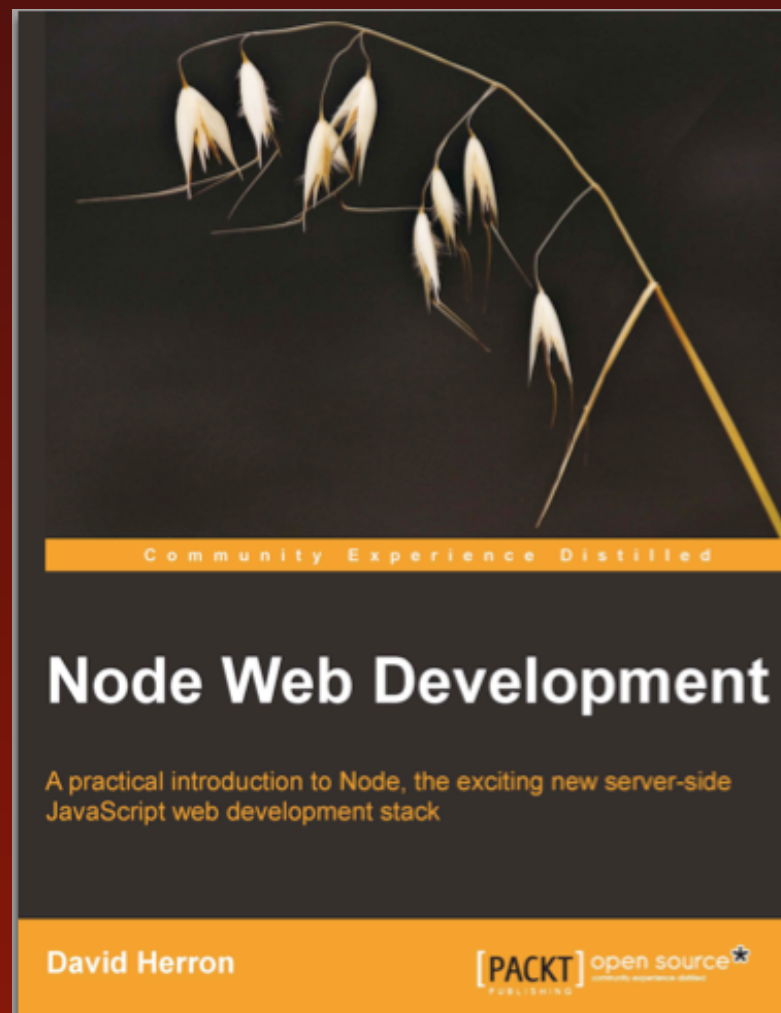
HANDS - ON

NODE.JS

THE NODE.JS INTRODUCTION AND API REFERENCE
BY PEDRO TEIXEIRA

node.js books

Node Web Development (Packt)



not so good!

node.js Hosting

Joyent (no.de)

Nodejitsu

Nodester

Heroku

easy; cheap;

Thanks for listening!