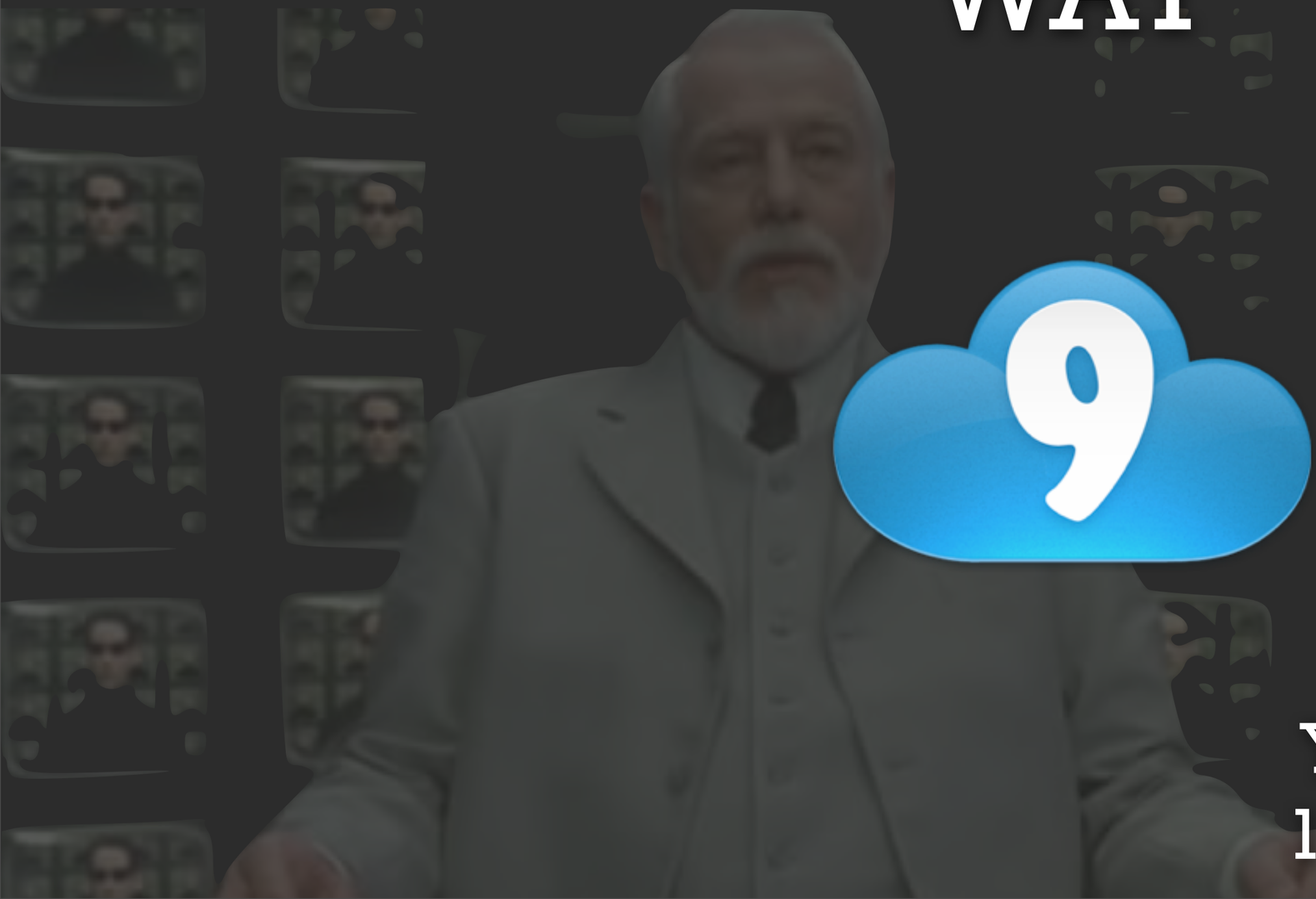
# THE ARCHITECT WAY





Jan Jongboom

Yet Another Conference

1 October 2012, Moscow


@drbernhard


9
{JS

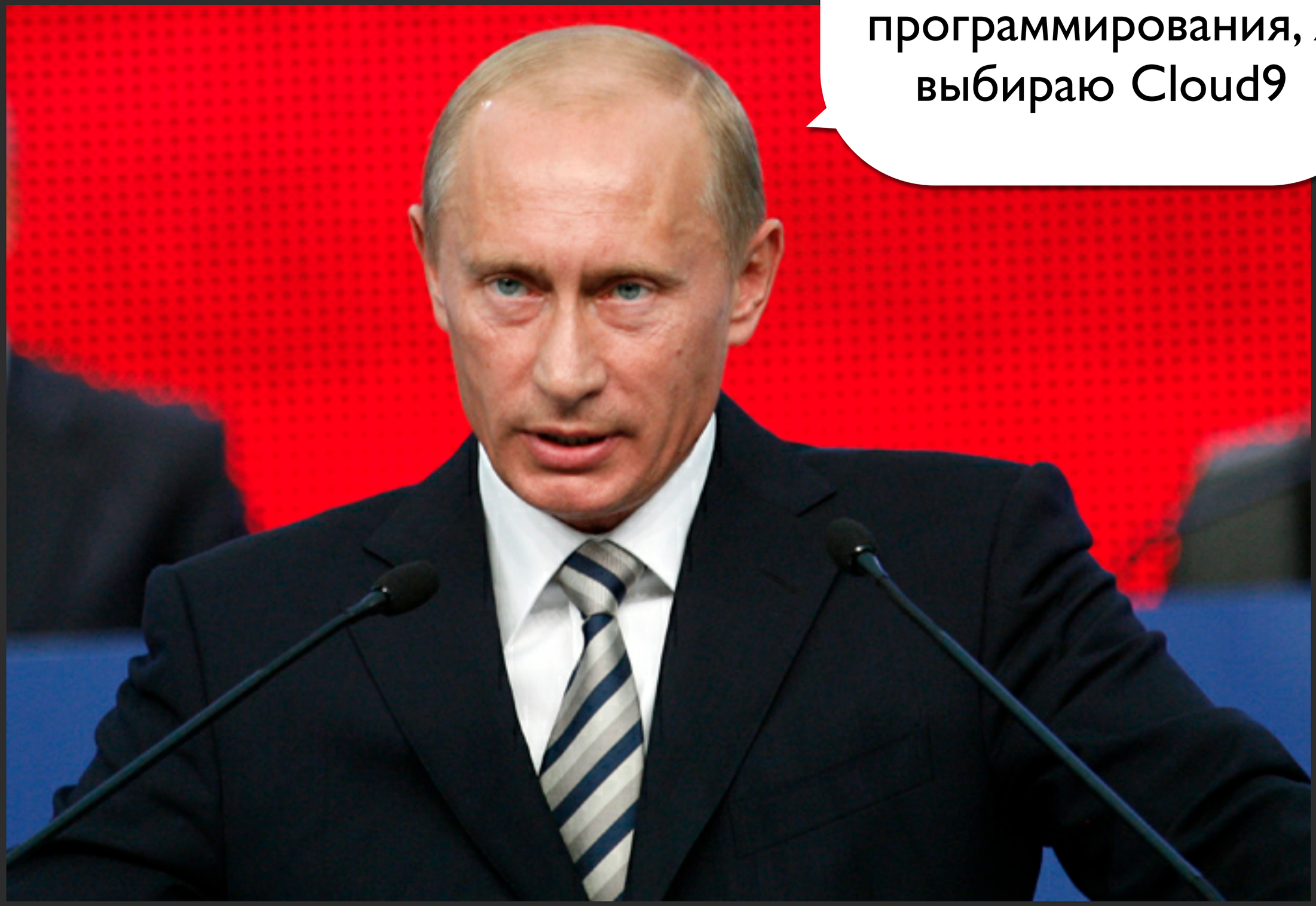

re("assert");
= function (options, imports, regis
ual(typeof options.numberOfDances, "nu
MakePresentation(presentor) {
ntor.dance()
ntor.speak("javascript")
var i = 0; i < options.numberOfDances;
esentor.dance()
{JS

# Program

- Cloud9? 5 minute intro + what's new
- Problems growing your codebase
- Introducing: Architect!
- Lessons learned

# Normal developers

# JavaScript Developer


Для динамического программирования, я выбираю Cloud9

File Edit Selection Find View Goto Tools Help Preview debug CHANGES SAVED AVAILABLE ONLINE OFF ON

draft.md presentation.js architect.js

PROJECT FILES

- yac2012
  - node_modules
    - architect
      - demos
      - plugins
      - architect.js
      - LICENSE
      - package.json
      - README.markdown
    - presenter
      - plugins
        - dance-reactor
          - package.json
          - react-on-dance.js
        - do-presentation
          - package.json
          - presentation.js
          - presentation_test.js
        - eventbus
          - eventbus.js
          - package.json
        - jan
          - jan.js
          - package.json
    - app.js
  - webserver
    - plugins
      - webserver

```
115     }
116
117     return sorted;
118 }
119
120 function Architect(config) {
121     var app = this;
122     app.config = config;
123     var services = app.services = {
124         hub: {
125             on: function (name, callback) {
126                 app.on(name, callback);
127             }
128         }
129     };
130
131     // Check the config
132     try {
133         var sortedPlugins = checkConfig(config)
134     } catch (err) {
135         return app.emit("error", err);
136     }
137
138     var destructors = [];
139
140     function startPlugins() {
141         var plugin = sortedPlugins.shift();
142         if (!plugin)
143             return app.emit("ready", app);
144
145         var imports = {};
146         if (plugin.consumes) {
147             plugin.consumes.forEach(function (name) {
148                 imports[name] = services[name];
```

15:20

Type "help" to get a list of commands

yac2012 - Cloud9 Local

localhost:13339/janjongboom/a87d2b63

File Edit Selection Find View Goto Tools Help Preview debug CHANGES SAVED AVAILABLE ONLINE OFF ON

draft.md presentation.js architect.js

PROJECT FILES

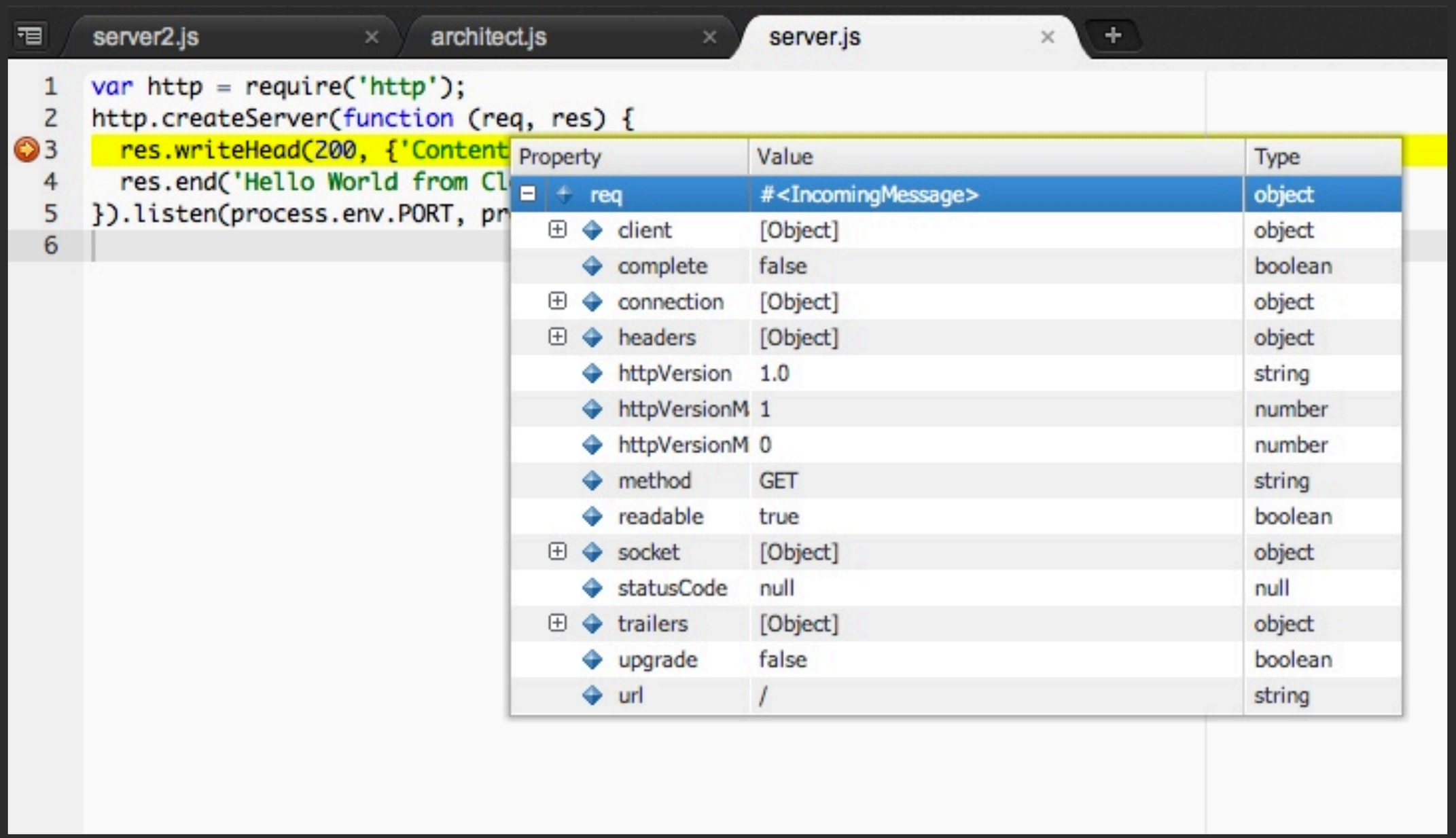
- yac2012
  - node_modules
    - architect
      - demos
      - plugins
      - architect.js
      - LICENSE
      - package.json
      - README.markdown
    - presenter
      - plugins
        - dance-reactor
          - package.json
          - react-on-dance.js
        - do-presentation
          - package.json
          - presentation.js
          - presentation_test.js
        - eventbus
          - eventbus.js
          - package.json
        - jan
          - jan.js
          - package.json
    - app.js
  - webserver
    - plugins
      - webserver

```
115     }
116
117     return sorted;
118 }
119
120 function Architect(config) {
121     var app = this;
122     app.config = config;
123     var services = app.services = {
124         hub: {
125             on: function (name, callback) {
126                 app.on(name, callback);
127             }
128         }
129     };
130
131     // Check the config
132     try {
133         var sortedPlugins = checkConfig(config)
134     } catch (err) {
135         return app.emit("error", err);
136     }
137
138     var destructors = [];
139
140     function startPlugins() {
141         var plugin = sortedPlugins.shift();
142         if (!plugin)
143             return app.emit("ready", app);
144
145         var imports = {};
146         if (plugin.consumes) {
147             plugin.consumes.forEach(function (name) {
148                 imports[name] = services[name];
```

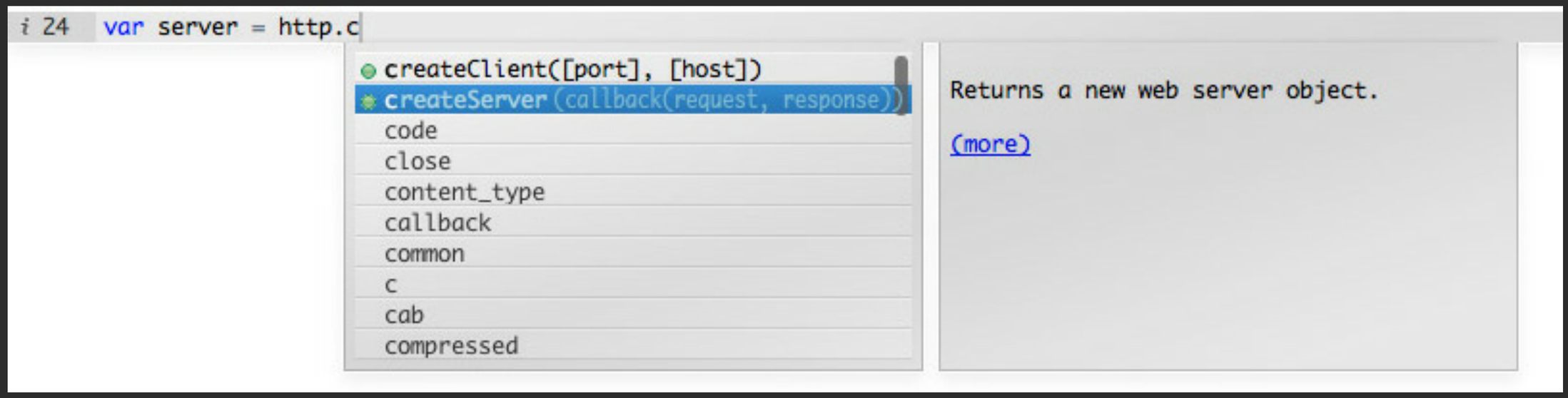
15:20

Type "help" to get a list of commands

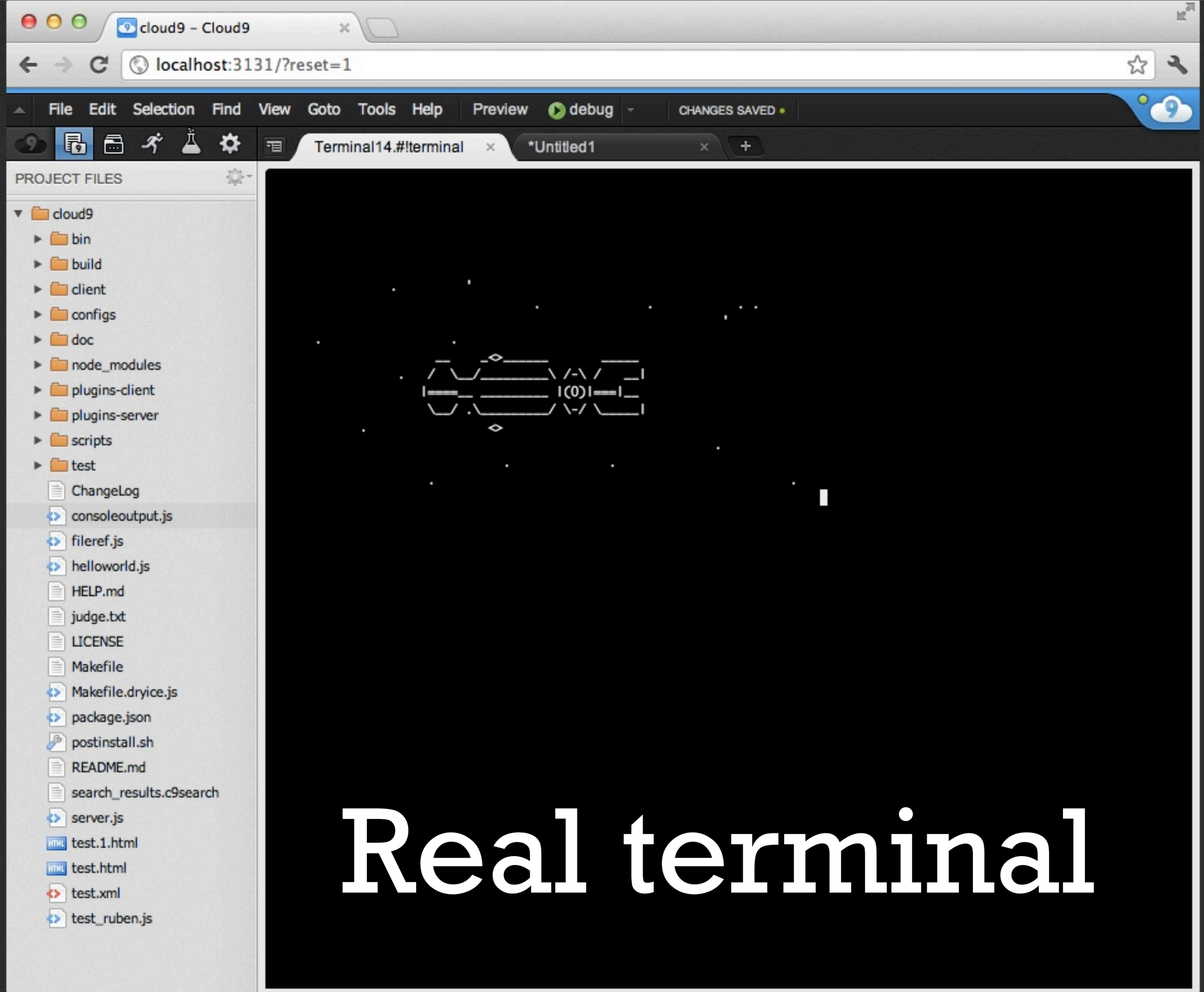
# Debugging


server2.js
architect.js
server.js
1 var http = require('http');
2 http.createServer(function (req, res) {
3 res.writeHead(200, {'Content
4 res.end('Hello World from Cl
5 }).listen(process.env.PORT, pr
6
Property
Value
Type
req
#<IncomingMessage>
object
client
[Object]
object
complete
false
boolean
connection
[Object]
object
headers
[Object]
object
httpVersion
1.0
string
httpVersionM
1
number
httpVersionM
0
number
method
GET
string
readable
true
boolean
socket
[Object]
object
statusCode
null
null
trailers
[Object]
object
upgrade
false
boolean
url
/
string

# (Smart!) Code completion




Free Linux VM!


cloud9 - Cloud9
localhost:3131/?reset=1
File Edit Selection Find View Goto Tools Help Preview debug CHANGES SAVED
Terminal14.#!terminal
*Untitled1
PROJECT FILES
cloud9
bin
build
client
configs
doc
node_modules
plugins-client
plugins-server
scripts
test
ChangeLog
consoleoutput.js
fileref.js
helloworld.js
HELP.md
judge.txt
LICENSE
Makefile
Makefile.dryice.js
package.json
postinstall.sh
README.md
search_results.c9search
server.js
test.1.html
test.html
test.xml
test_ruben.js
Real terminal

# Collaboration

See each other type

Debug together

Productivity++

# Program

- Cloud9? 5 minute intro + what's new
- Problems growing your codebase
- Introducing: Architect!
- Lessons learned

# Pure madness

Openshift

Cloud9 datacenter

IDE server

Remote VM

Remote VM

Remote VM

# Rules of Jan

- A codebase needs modularization
- Modularization abstracts features away
- Teams can work on separate features without breaking code

# Modularization?

- Node.js has require
- Similar to ‘using’ or ‘import’ in .NET / Java
- Great for abstracting away functionality
- But not for application modularity

```
var db = require("database");
db.doStuff();
```

# Downsides of require

- Relies on the filesystem
  - Two modules, same dependencies?
    Copy or symlink
  - Maps to folder name
- Configuration is hard
- Multiple instances

# No dependency model

- Static compilation: build dependency tree on compile time
- Dependency tree not good? Compilation error!
- Require fails at runtime

# Now fix it!

- Static dependency list
  - Resolve at startup
- Named services
  - No longer require filesystem
- Easy configuration options

# Program

- Cloud9? 5 minute intro + what's new
- Problems growing your codebase
- Introducing: Architect!
- Lessons learned

# Architect

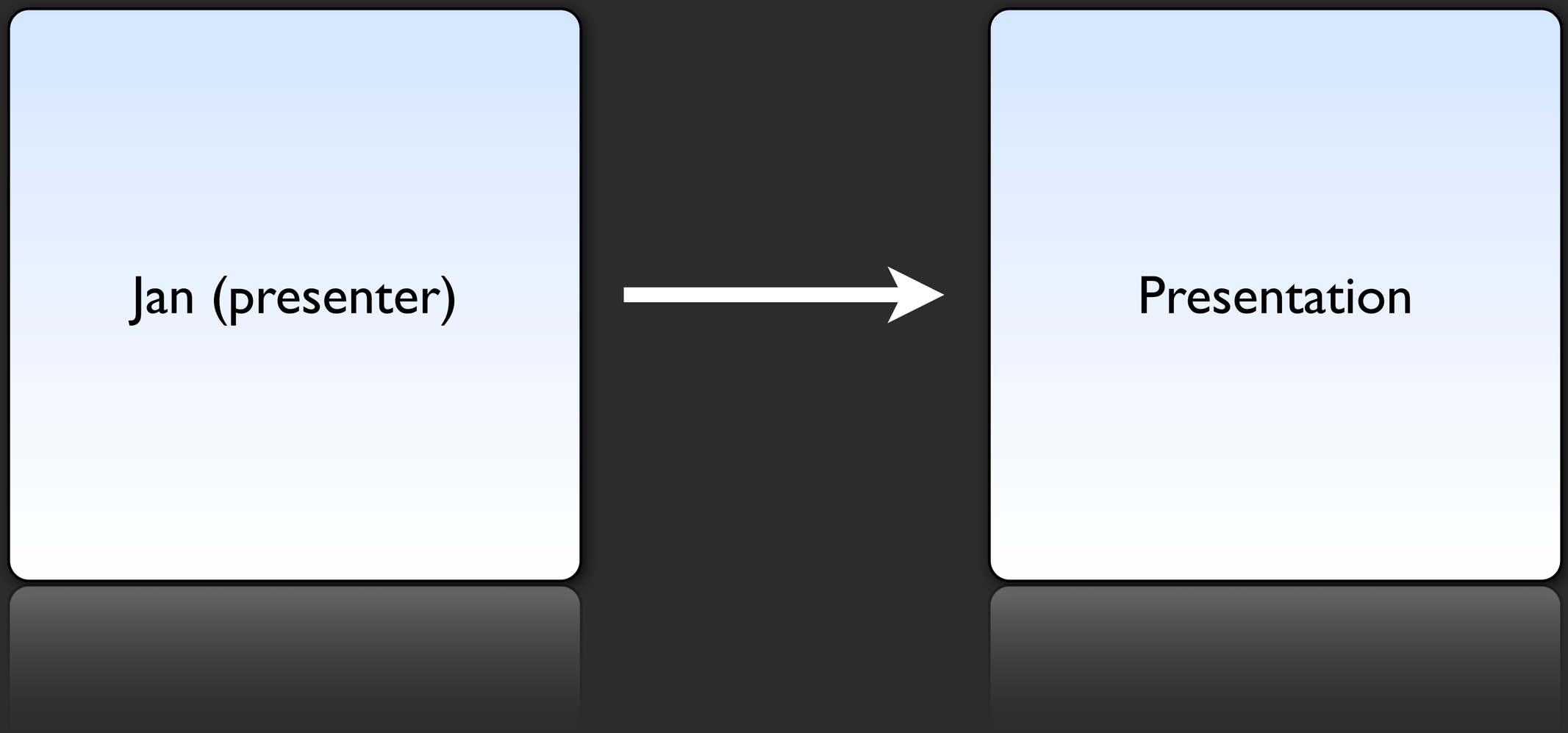
- Every piece of functionality is a plugin
- Plugins can consume other plugins
- An application is a set of plugins

```
function doPresentation () {
    var jan = {
        dance: function () {
            /* implementation can be seen at the afterparty */
        },
        speak: function (subject) {
            console.log("blah blah blah")
        }
    };

    jan.dance()
    jan.speak("javascript")

    for (var i = 0; i < 10; i++)
        jan.dance()
}
```

- Declare entity ‘jan’ with behavior
- Use ‘jan’ to do a presentation

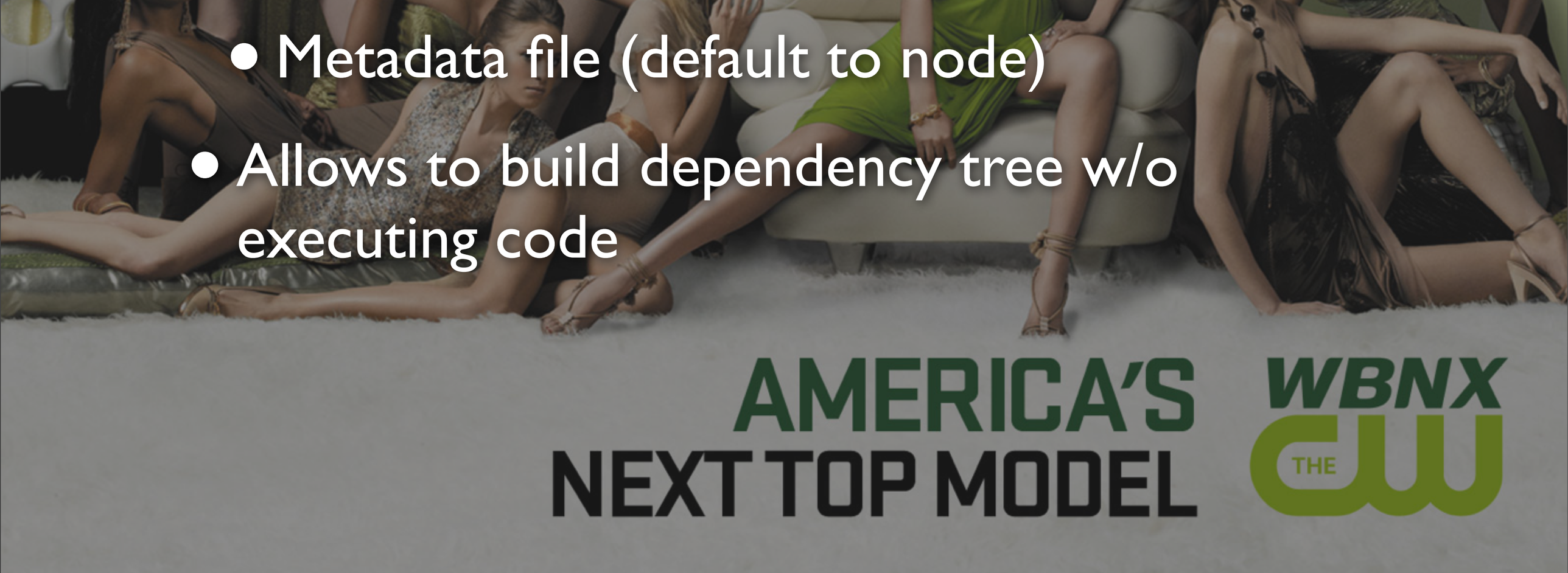
# Group functions by behavior

# Dependency model



# How to express our model

- Package.json
  - Metadata file (default to node)
  - Allows to build dependency tree w/o executing code



```
// jan/package.json
{
    "name": "jan",
    "main": "./jan.js",

    "plugin": {
        "consumes": [],
        "provides": [
            "presenter"
        ]
    }
}
```

```
// do-presentation/package.json
{
    "name": "presentation",
    "main": "./presentation.js",

    "plugin": {
        "consumes": [ "presenter" ],
        "provides": []
    }
}
```

# What's next?

- Extract the code
- Wrap in Architect plugin code
  - It's simple!
- Make two plugins

Function signature

```
module.exports = function (options, imports, register) {
    var jan = {
        dance: function () {
            /* wait for the afterparty! */
        },
        speak: function (subject) {
            console.log("blah blah blah")
        }
    };

    register(null, {
        "presenter": jan
    })
}
```

Call when done

# Architect plugin code

- Module.exports
- Options, will get to that
- Imports, everything you ‘consume’
- Register, invoke when done

```
module.exports = function (options, imports, register) {
    var presenter = imports.presenter

    presenter.dance()
    presenter.speak("javascript")

    for (var i = 0; i < 10; i++)
        presenter.dance()

    // nothing to provide
    register()
}
```

# Dependencies abstracted away

- Easily unit testable
- Mock dependencies
- Assert ‘dance’ function is called 11 times

```
// just reference the plugin (no architect required)
var doPresentation = require("./presentation");

// when calling this we can mock it
var mockedJan = {
    dance: createStub(),
    speak: createStub()
};

var options = {};
var imports = {
    "presenter": mockedJan
};

doPresentation(options, imports, function () {
    // assert we called the dance function 11 times
    assert.equal(mockedJan.dance.callCount, 11);
});
```

# No black magic

- Dependency model needs to be specified
- Feed Architect a config file
  - Simple array with list of plugins
- Call ‘createApp’

```
// app configuration
var config = [
    {
        packagePath: "./plugins/jan"
    },
    {
        packagePath: "./plugins/do-presentation"
    }
];

// create relative tree
var tree = architect.resolveConfig(config, __dirname);

// start app
architect.createApp(tree, function () {
    console.log("Application started");
});
```

```
Could not resolve dependencies of these plugins:

[ { packagePath: '/plugins/do-presentation/package.json',
  provides: [],
  consumes: [ 'presentation' ],
```

# Configuration

- Per-plugin options
- No global options object
- Specify in config file

```
module.exports = function (options, imports, register) {
```

```
    {
        packagePath: "./plugins/do-presentation",
        numberOfDances: 8
    }
```

# Options

- Automatically passed in at startup
- Options are also dependencies
- Fail if options aren't present
  - Use default assertions

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

module.exports = function (options, imports, register) {
    // you can also do type assertion here, check if it's a number etc.
    assert(options.numberOfDances, "Option 'numberOfDances' is required");

    /* snip some code */

    // usage
    for (var i = 0; i < options.numberOfDances; i++)
        jan.dance();

    register(null, null);
};
```

```
AssertionError: Option 'numberOfDances' is required
    at Object.setup (/plugins/do-presentation/presentation.js:4:5)
```

Architect makes you think of
your app as
chunks of functionality
rather than sets of classes

# Think ‘chunks of functionality’

- Implicit type constraints
  - Keep implementation private
- Swap feature implementations
  - Rather than interface implementations

# How do we use it?

- Open source version
- Local version (OS + sync)
- Hosted version
  - Normal
  - FTP
  - SSH

```
var fs = imports.fs;

function saveFile (filename, contents) {
    fs.writeFile(filename, contents);
}
```

# Swap feature per implementation

- On Open source: talk local filesystem
- On FTP: talk FTP library
- On SSH: log in and talk via a SSH bridge

Here is something

your DI framework

can’t do

Single node.js process

Other code
(dashboard etc.)

IDE instance
(FTP)

IDE instance
(SSH)

IDE instance
(Normal)

IDE instance
(Normal)

```
// let's assume we're making a multi player online game
function getGameStatus(req) {
    var gameId = req.query.gameId
    var game = db.GetGame(gameId)

    return game.getStatus()
}
```

# Architect can do

- Multiple instances of same plugin
- Run independent
  - In separate contexts
- But in the same process
  - Manageable
  - No process overhead

```
function startNewGame (name) {
    var game = db.createNewGame(name);

    var config = [
        {
            packagePath: "./game-status",
            game: game
        }
];

architect.createApp(/* snip */
```

```
// game-status.js
module.exports = function (options, imports, register) {
    assert(options.game, "Option 'game' is required");

    // even cache stuff in this specific context
    var localState = {};

    // define base url in init code (one time executed!)
    var baseUrl = "/" + options.game.name;

    // when request comes in
    http.get(baseUrl + "/status", function () {
        // no context switching!
        res.send(game.getStatus());
    });
};
```


80's 1337ness
CAPCOM
DuckTales
CAPCOM
VIPER
NINTENDO POWER
CAPCOM
HERE'S SOMETHING
COOL
wildelec.org

# Centralized eventbus

- Loose coupling between plugins
- No hard dependencies!
- Can also do inter-context communication

Other plugin

React on event

Eventbus

Emit event

Plugin

```
var EventEmitter = require("events").EventEmitter;

module.exports = function (options, imports, register) {
    var emitter = new EventEmitter();

    register(null, {
        "eventbus": {
            emit: emitter.emit,
            on: emitter.on
        }
    });
}
```

```
var bus = imports.eventbus;

// every time a presenter makes a dance
bus.emit("dancing");

// react on events in other plugins
// see the dance-reactor in the repo
bus.on("dancing", function () {
    console.log("Someone is dancing!");
});
```

# And now scale up

- Need something inter-server
- Swap it with i.e. Redis PubSub
- Plugins will never notice
- Awesome!

# Program

- Cloud9? 5 minute intro + what's new
- Problems growing your codebase
- Introducing: Architect!
- Lessons learned

# Modularize in feature blocks

- Don’t over engineer
- Don’t create too small blocks
  - They are no interfaces!

# Use dependency injection

- Architect (javascript)
- StructureMap (.NET)
- Spring (Java)

# Avoid context switching

- Less code!
- Less errors!
- Less boilerplate!
- StructureMap has some basics for this as well

# Loose coupling

- Eventbus
- Smaller dependency graph

# Cloud9 IDE

*Your code anywhere, anytime*

github.com/c9/yac2012

github.com/c9/architect

Happy coding!

# Cloud9 IDE

*Your code anywhere, anytime*

http://c9.io

Jan Jongboom
github.com/janjongboom
@drbernhard