



9

Architecting large node.js applications

@sergimansilla

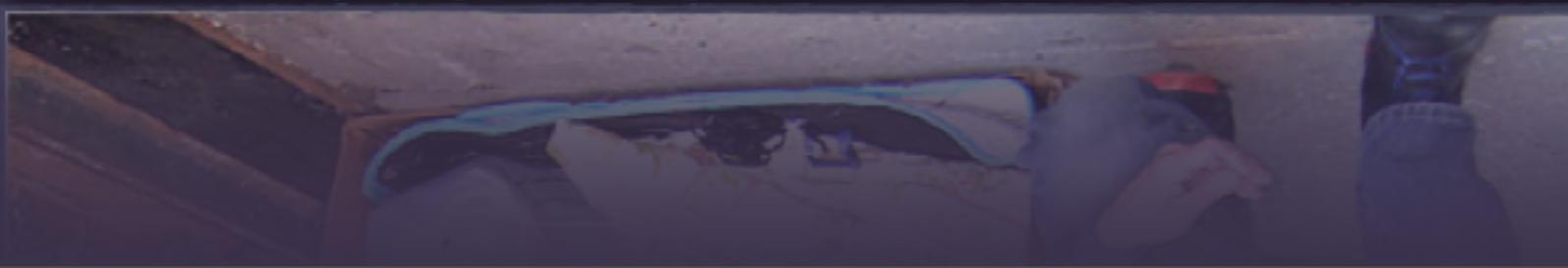
Program

- Cloud9 IDE?
- Growing pains
- Introducing Architect
- Lessons learned

Normal developers



JavaScript Developer



Cloud9 IDE - <http://c9.io>

The screenshot shows the Cloud9 IDE interface. The main area is a code editor with the file `revisions.js` open. The code is a JavaScript module for a Revisions Server. A tooltip is visible over the word `createEmptyStack`, showing its definition in another part of the code. The project tree on the left shows the directory structure of the Cloud9 client programs. The top bar includes standard menu items like File, Edit, Selection, Find, View, Goto, Tools, Share, Help, and a Debug button. On the right, there are toolbars for preview, availability, and a call stack window which is currently empty.

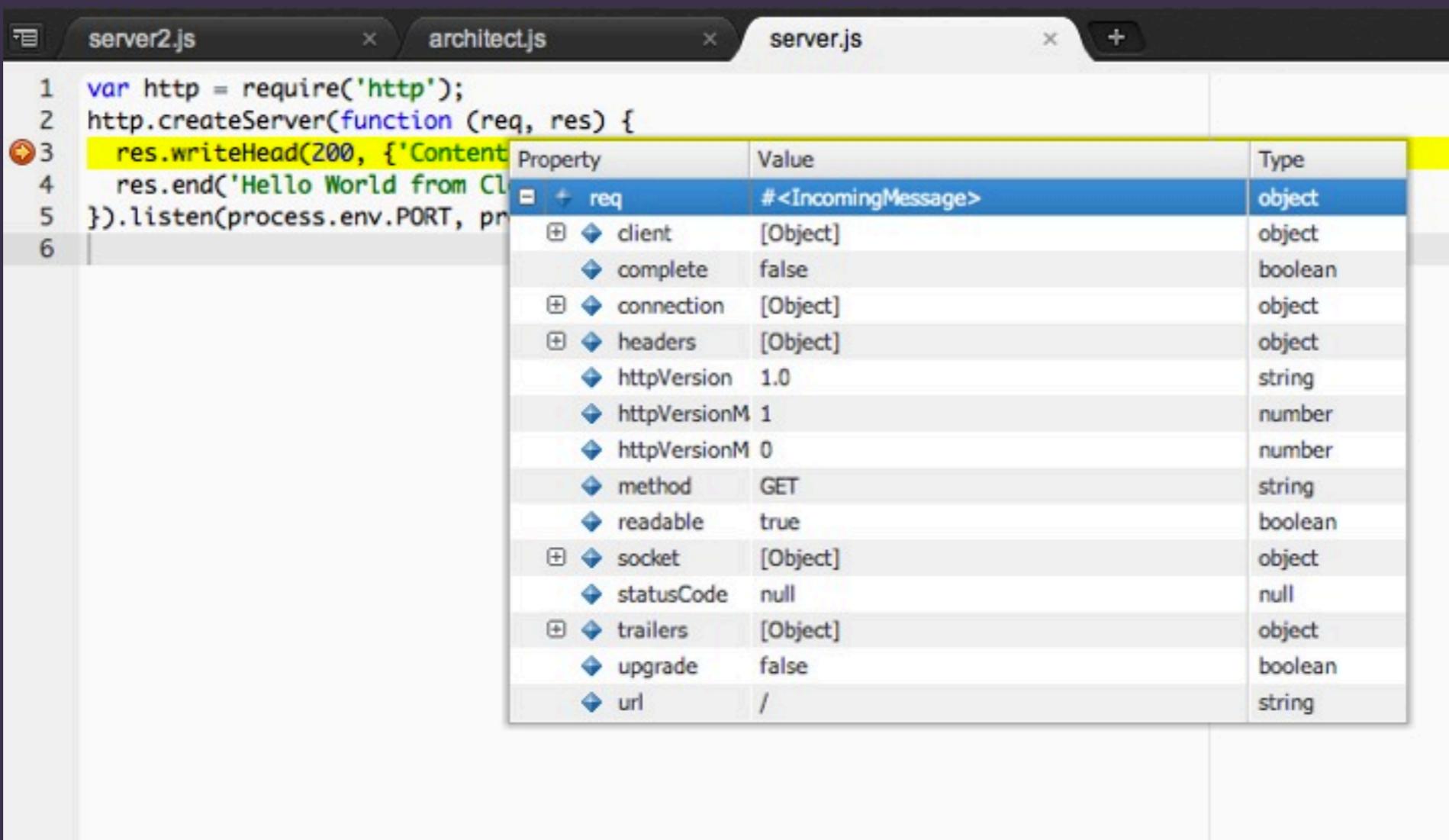
```
1 /**
2  * Revisions Server module for Cloud9 IDE
3  *
4  * @author Sergi Mansilla <sergi@c9.io>
5  * @copyright 2012, Ajax.org B.V.
6 */
7
8 require("amd-loader");
9 var Plugin = require("../cloud9.core/plugin");
10 var Diff_Match_Patch = require("./diff_match_patch");
11 var Path = require("path");
12 var PathUtils = require("./path_utils.js");
13 var Async = require("async");
14 var fsnode = require("vfs-nodefs-adapter");
15
16 /* */
17 var FILE_SUFFIX = "c9save";
18
19 /* */
20 var REV_FOLDER_NAME = ".c9revisions";
21
22 var Diff = new Diff_Match_Patch();
23 var name = "revisions";
24
25 module.exports = function setup(options, imports, register) {
26     var fs;
27
28     function createEmptyStack() fn
29         Plug
30         createEmptyStack
31         create
32         this
33         created
34         this.cr
35
36         var _self = this;
37         // This queue makes sure that changes are saved asynchronously but orderly
38
39         this.createEmptyStack()
40         this.created()
41         this.createEmptyStack()
42
43         var _self = this;
44         // This queue makes sure that changes are saved asynchronously but orderly
```

copy

Monday, October 22, 12

Open a Terminal

Real debugging



A screenshot of a debugger interface showing the properties of the 'req' object. The code in the editor shows a snippet of Node.js code creating an HTTP server. A breakpoint is set at line 3, where 'res.writeHead' is called. A tooltip-like window displays the structure of the 'req' object.

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

```
1 var http = require("http");
2 http.createServer(function(req, res) {
3     res.writeHead(200, {'Content-Type': 'text/plain'});
4
5     res.
6         ● addTrailers(headers)
7         ● end([data], [encoding])
8         ● getHeader(name)
9         ● removeHeader(name)
10        ● setHeader(name, value)
11        ● statusCode
12        ● write(chunk, [encoding])
13        ● writeContinue()
14        ● writeHead(statusCode, [reasonPhrase], [headers])
15
16
17
18
19 }).listen(process.env.PORT, process.env.IP);
20
```

[getHeader\(name\) : String](#)

Reads out a header that's already been queued but not sent to the client. Note that the name is case-insensitive. This can only be called before headers get implicitly flushed.

[\(more\)](#)

Static analysis

```
1
2 var ASSERT = require("assert");
3 var PATH = require("path");
4 var SYNCER = require("c9-vfs-sync/lib/syncer");
5 var CMD_WRAPPER = require("c9-vfs-sync/lib/cmd-wrapper");
6 var UTIL = require("util");
7 var HTTP_ERROR = require("http-error");
8 var FS_NODE = require("vfs-nodefs-adapter");
i 9 var fs = require("fs");
10
11 module.exports = function setup(options, imports, register) {
12
*x 13     ASSERT(typeof options.enabled !== "undefined" "options.enabled is required")
14     Expected ')' and instead saw 'options.enabled is
15     required'. Missing semicolon. Expected an
16     identifier and instead saw ')'.
17
18     var api = {
i 19         on: function(e) {
20             if (!syncer) return;
21
22             this.next();
23             syncer.on.apply(syncer, arguments);
24             syncer.const = "test";
25         },

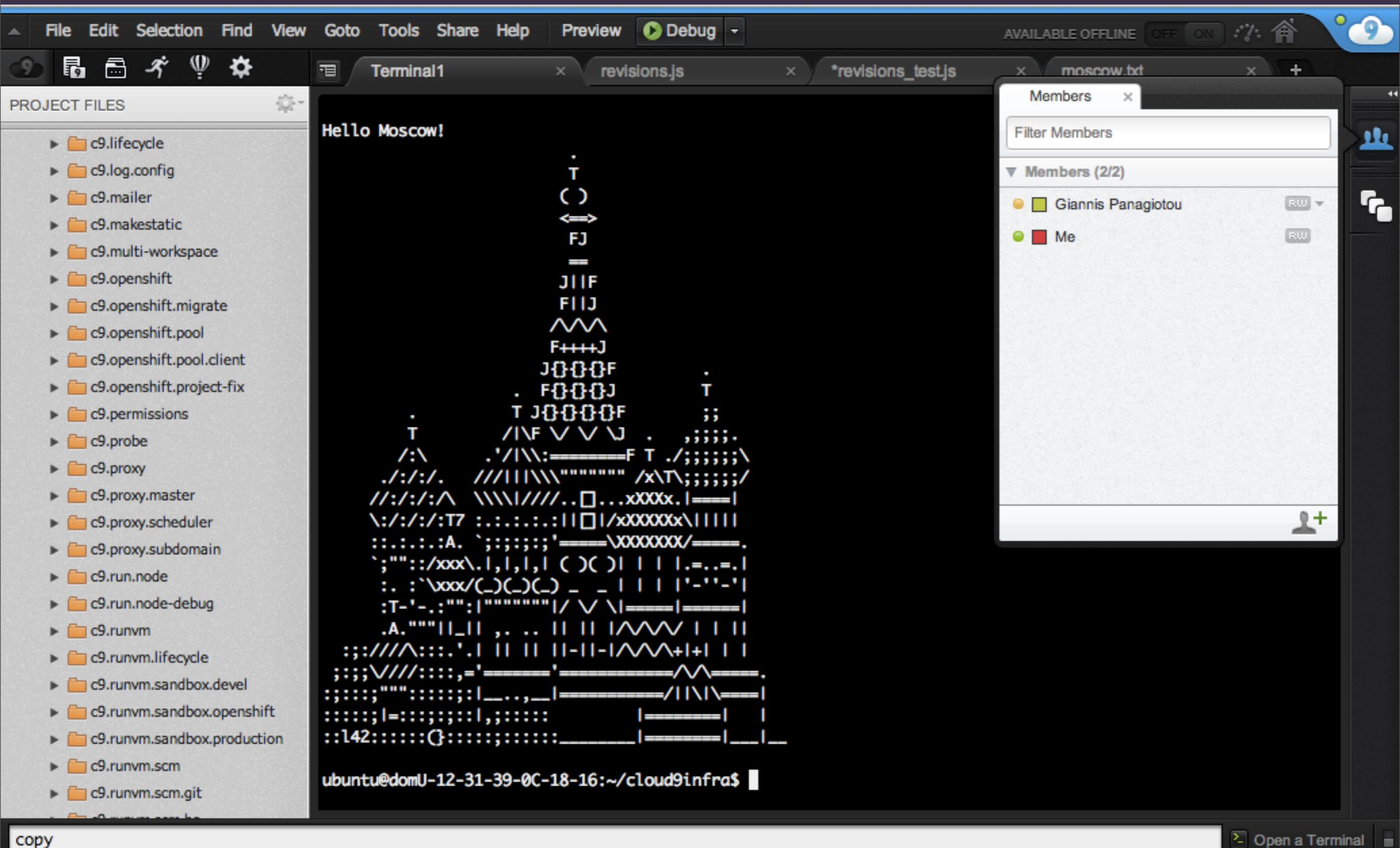
```

Free
Linux
VM

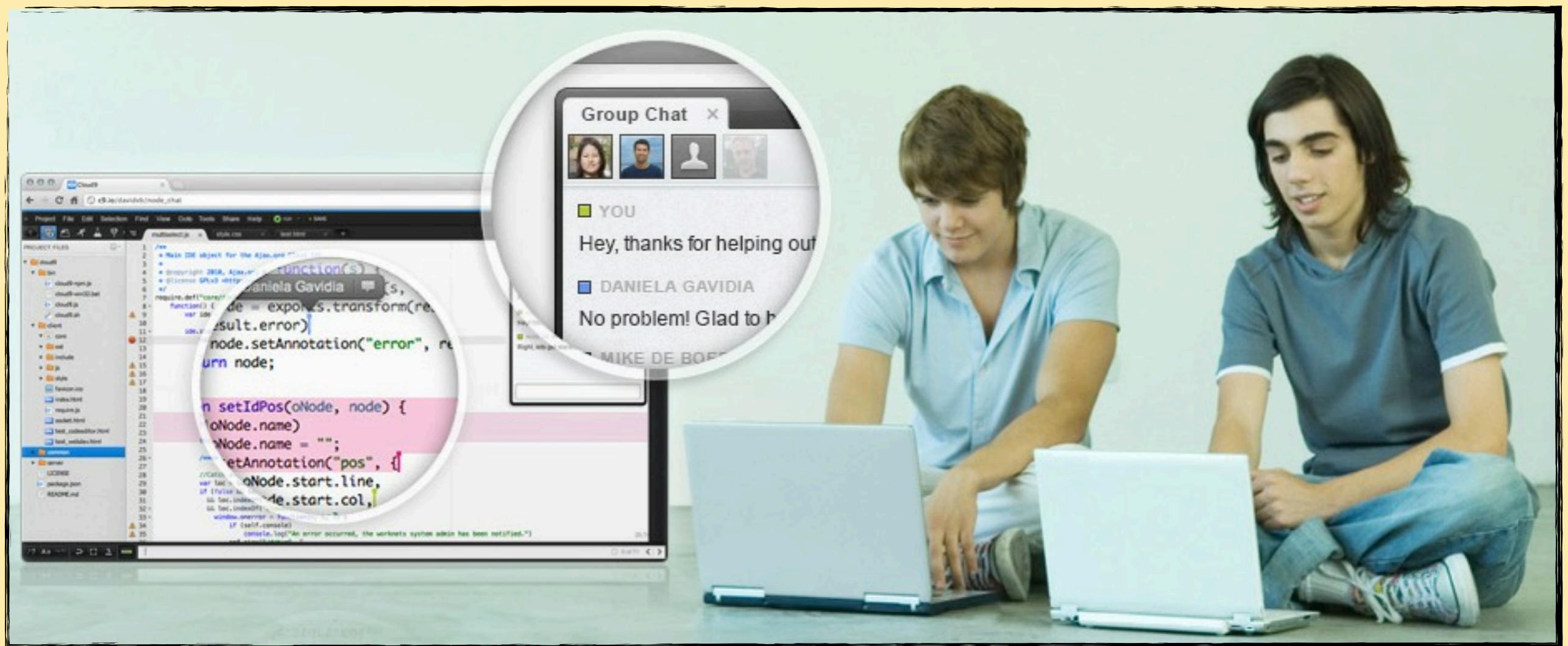


Bring your own machine

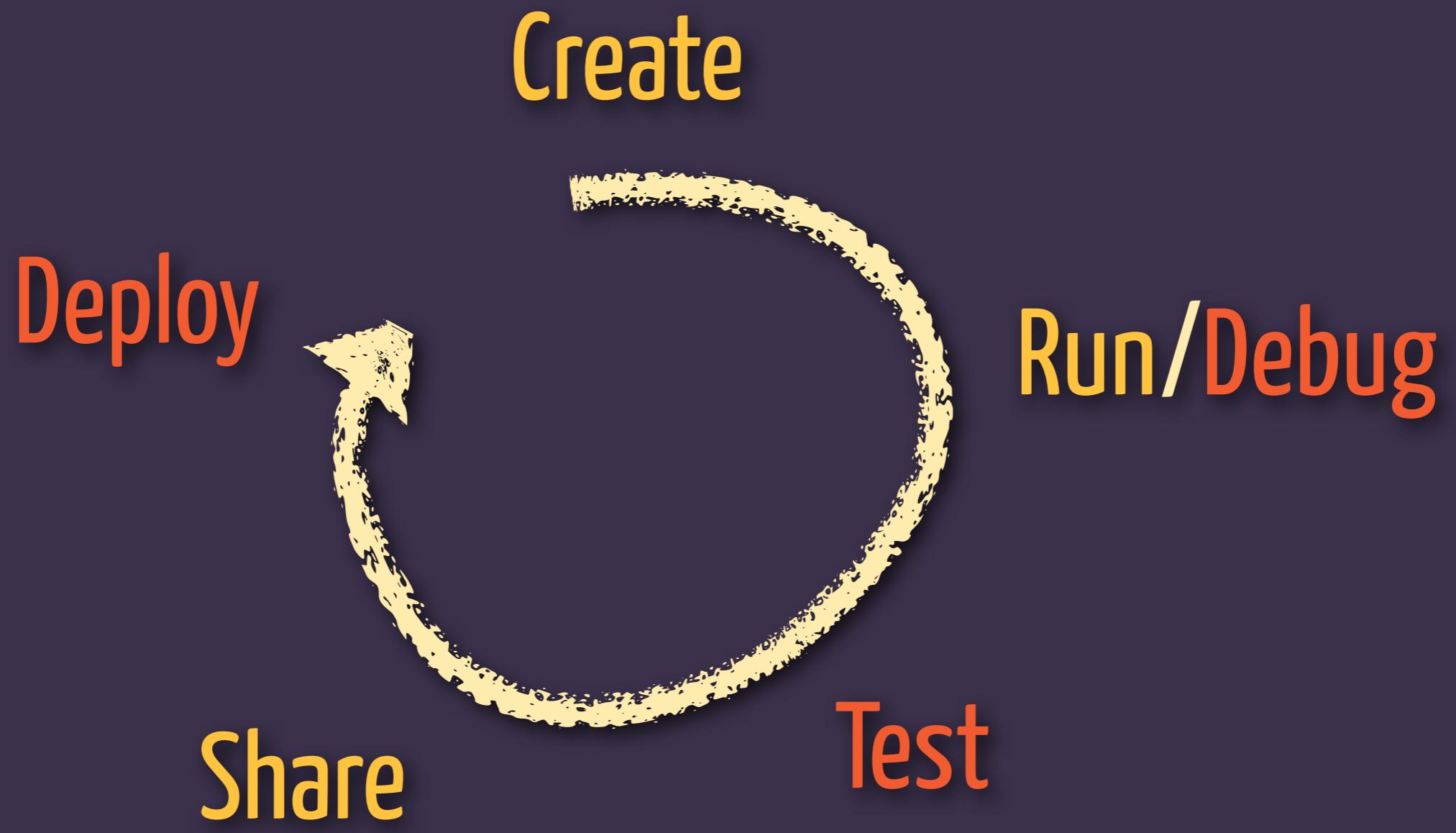
Real terminal



Collaboration



See each other type
Debug together
Productivity++



Program

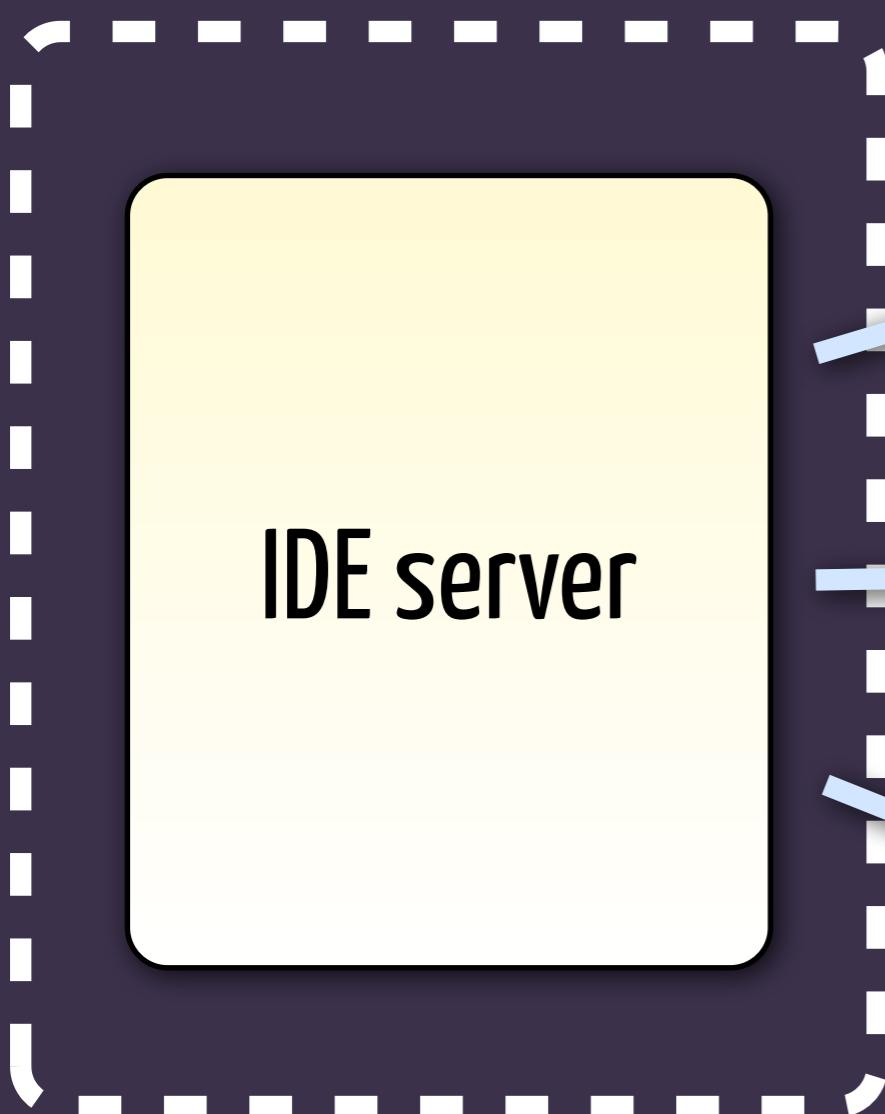
- Cloud9 IDE?
- Growing pains
- Introducing Architect
- Lessons learned

10,000s
LOC
of JavaScript

**dynamic,
weakly
typed
language**

One
single
thread

Cloud9 datacenter

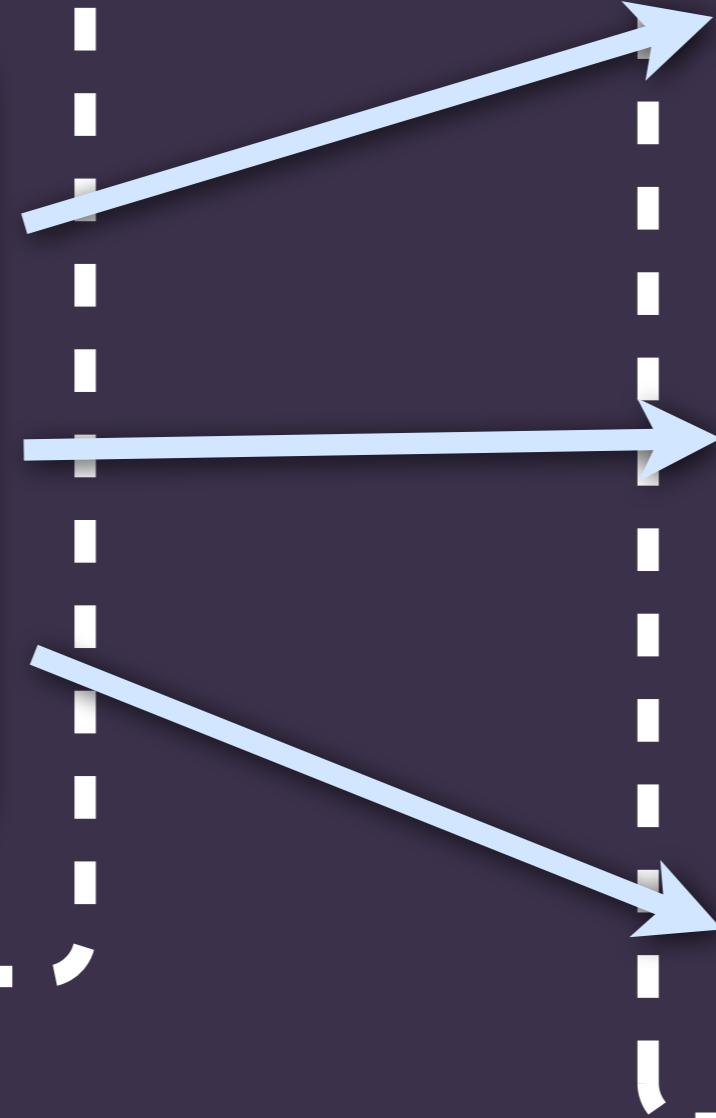


Openshift

Remote VM

Remote VM

Remote VM



Pure madness



DINE IN HELL!!
a chance to

I dunno, lol ↗_(°·o·)↖

Modularization

**Black
box
coding**

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

Java/.NET

→ import(s)

Node.js

→ require

require

Great for
abstracting
away

require

Not Great for
application
modularity

require

Relies on FS

duplicated modules

maps to folder names

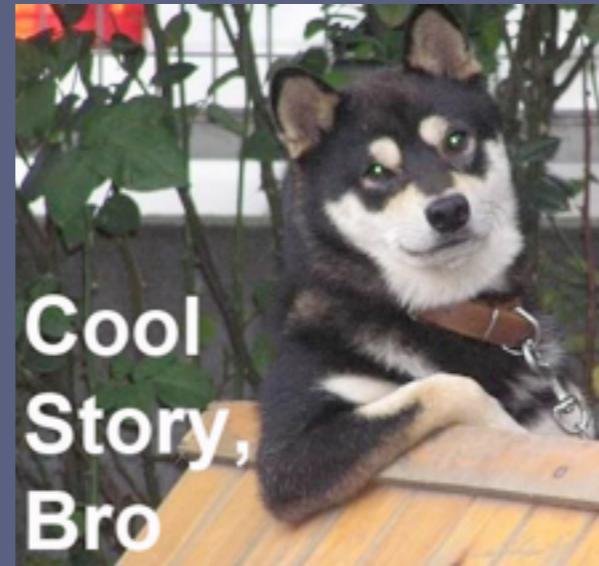
Hard to configure module

Dependency error handling

coding time



compile time



run time

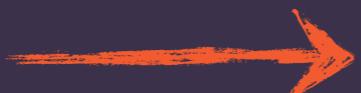


- Server crash
- Unhappy customers
- Developer gets fired



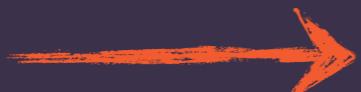
How to fix it?

Static dependency list



Resolve at startup

Named services



No FS required

Easy configuration



Pass an object

Program

- Cloud9 IDE?
- Growing pains
- Introducing Architect
- Lessons learned

Architect



github.com/c9/architect

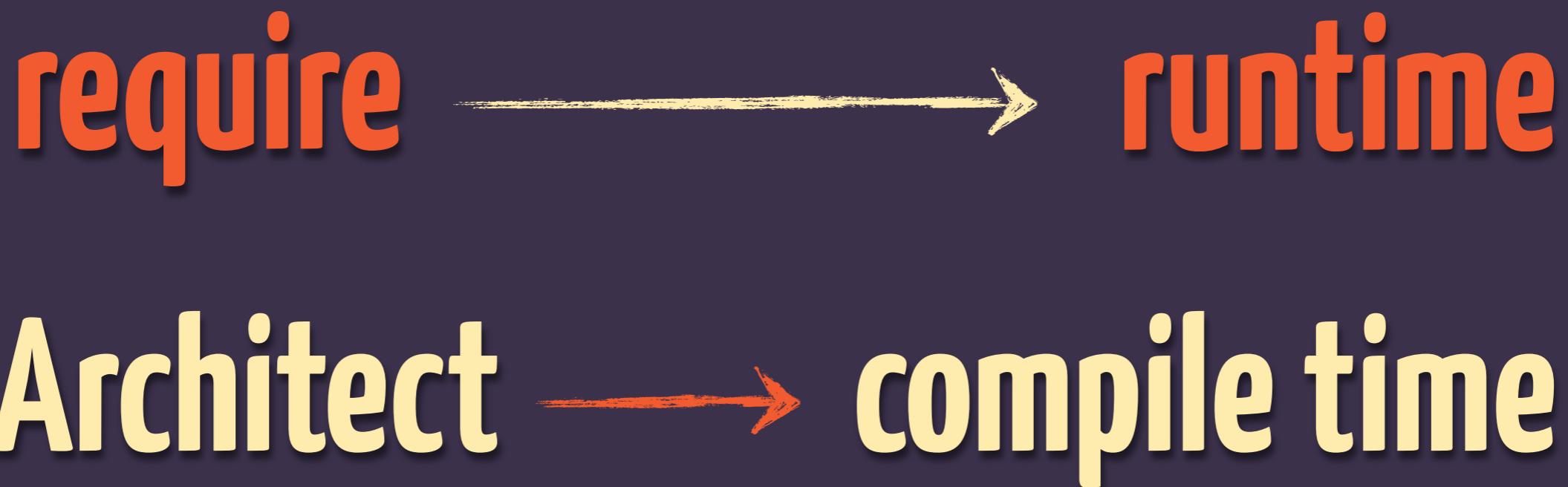
Architect

Everything is a plugin

Plugins can consume plugins

An application is just a set of plugins

Dependency model



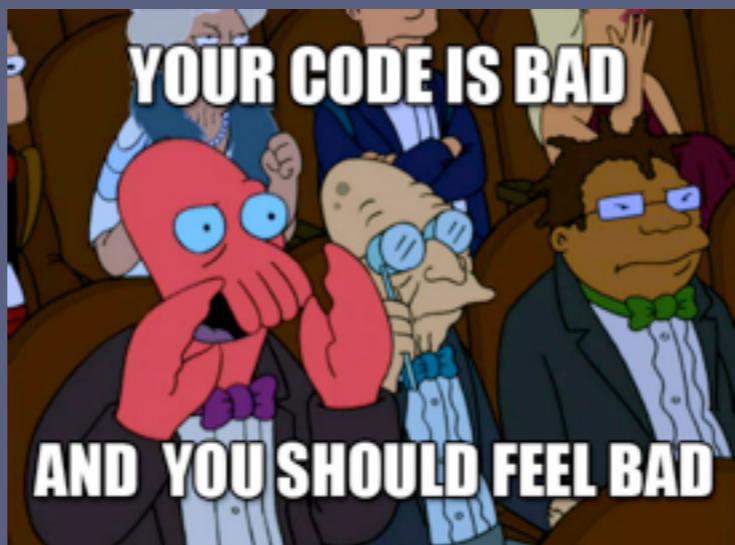
Dependency error handling

Architect

coding time

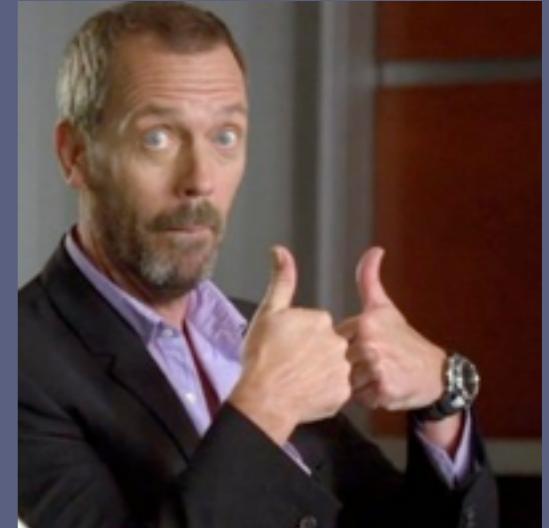


compile time

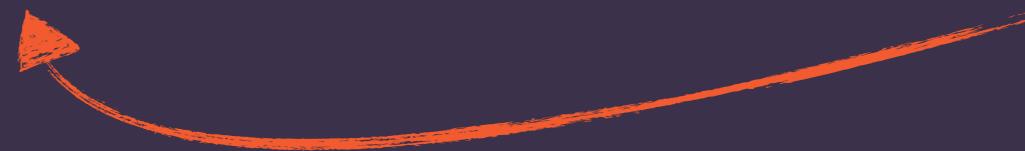


- Fails before release

run time



- Happy customers
- Developer keeps job

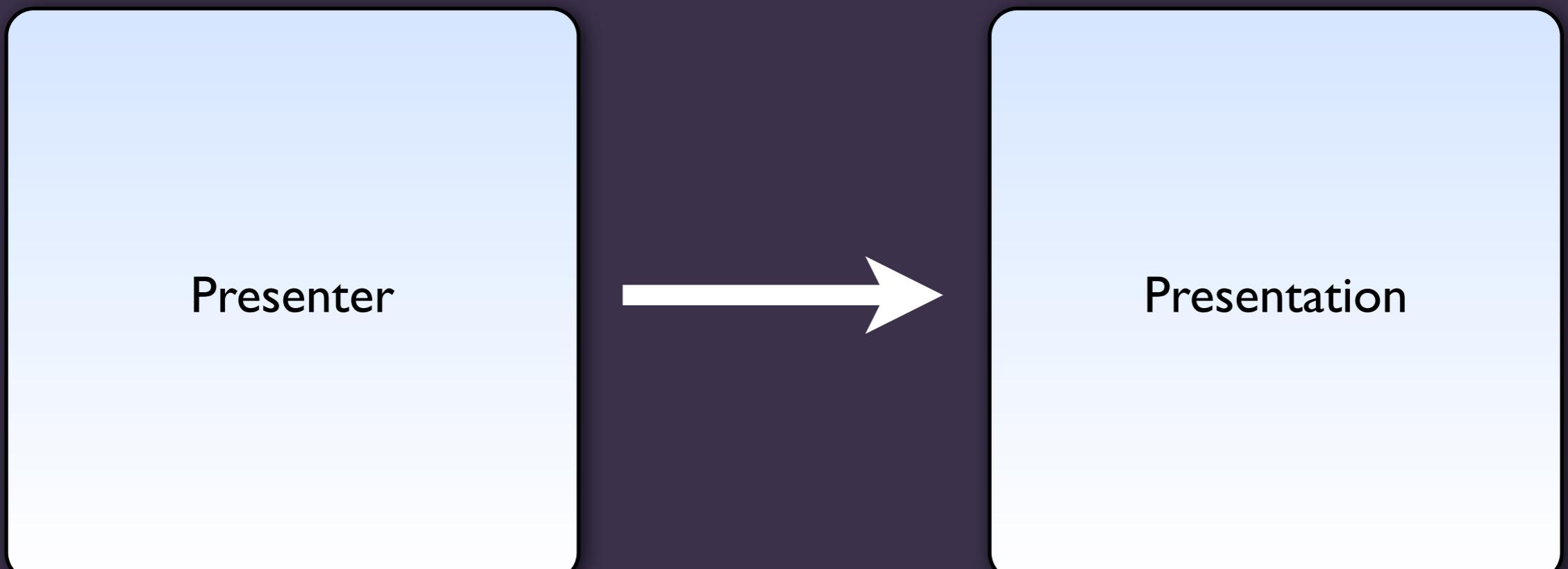


```
1 function doPresentation () {
2     var presenter = {
3         dance: function () {
4             /* implementation can be seen at the afterparty */
5         },
6         speak: function (subject) {
7             console.log("blah blah blah")
8         }
9     };
10
11     presenter.dance()
12     presenter.speak("javascript")
13
14     for (var i = 0; i < 10; i++)
15         presenter.dance()
16 }
```

Declare entity ‘presenter’ with behavior

Use ‘presenter’ to do a presentation

Dependency model



Express our model

package.json



Builds dependency tree
without
executing code

```
1 // presenter/package.json
2 {
3     "name": "presenter",
4     "main": "./presenter.js",
5
6     "plugin": {
7         "consumes": [],
8         "provides": [
9             "presenter"
10        ]
11    }
12 }
13
14 // do-presentation/package.json
15 {
16     "name": "presentation",
17     "main": "./presentation.js",
18
19     "plugin": {
20         "consumes": [ "presenter" ],
21         "provides": []
22     }
23 }
```

What's next?

Wrap in Architect plugin code

Extract the code

Make two plugins

Function signature

```
1 module.exports = function (options, imports, register) {  
2     var presenter = {  
3         dance: function () {  
4             /* wait for the afterparty! */  
5         },  
6         speak: function (subject) {  
7             console.log("blah " + subject + " blah")  
8         }  
9     };  
10    register(null, {  
11        "presenter": presenter  
12    })  
13}  
14 }
```

Call when done

Architect plugin code

Module.exports

Options - we'll 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()
}
```

```
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()
}
```



Easy to test
Mock dependencies

Assert ‘dance’ is called 11 times

```
1 // just reference the plugin (no architect required)
2 var doPresentation = require("./presentation");
3
4 // when calling this we can mock it
5 var mockedPresenter = {
6   dance: createStub(),
7   speak: createStub()
8 };
9
10 var options = {};
11 var imports = {
12   "presenter": mockedPresenter
13 };
14
15 doPresentation(options, imports, function () {
16   // assert we called the dance function 11 times
17   assert.equal(mockedPresenter.dance.callCount, 11);
18 });
```

No black magic

Specify dependency model

Feed architect a config file

Call 'createApp'

```
1 var config = [
2   {
3     packagePath: "./plugins/presenter"
4   },
5   {
6     packagePath: "./plugins/do-presentation"
7   }
8 ];
9
10 // Create relative tree
11 var tree = architect.resolveConfig(config, __dirname);
12
13 // Start app
14 architect.createApp(tree, function () {
15   console.log("Application started");
16 });
```

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

```
1 var assert = require("assert");
2
3 module.exports = function (options, imports, register) {
4     // you can also do type assertion here, check if it's a number etc.
5     assert(options.numberOfDances, "Option 'numberOfDances' is required");
6
7     /* snip some code */
8
9     // usage
10    for (var i = 0; i < options.numberOfDances; i++)
11        presenter.dance();
12
13    register(null, null);
14};
```

```
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 features instead of interfaces

How does Cloud9 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: talk via a SSH bridge

Here is something your
DI framework can't do

Single node.js process

The diagram illustrates a single Node.js process managing multiple components. A central horizontal dashed line represents the process boundary. On the left, a yellow rounded rectangle contains the text "Other code (dashboard etc.)". To the right of this, four white rounded rectangles with black borders represent different instances: "IDE instance (FTP)", "IDE instance (SSH)", "IDE instance (Normal)", and another "IDE instance (Normal)". All components are contained within the process boundary.

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 independently

But in the same process

```
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



HERE'S SOMETHING COOL



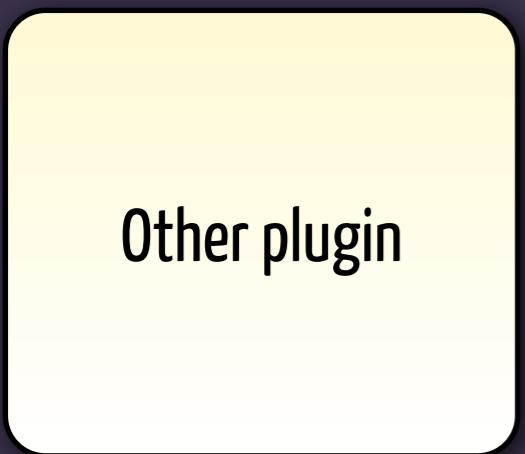
videlec.org

Centralized eventbus

Loose coupling between plugins

No hard dependencies!

Can also do inter-context communication



React on event

Other plugin

Event
bus

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

Modular awesomeness!



Program

- Cloud9 IDE?
- Growing pains
- Introducing Architect
- Lessons learned

Modularize in feature blocks

Don't over engineer

Don't create too small blocks

They aren't interfaces!

Use dependency injection

Architect (JavaScript)

StructureMap (.NET)

Spring (Java)

Avoid context switching

Less code!

Less errors!

Less boilerplate!

Loose coupling

Use an event bus

Smaller dependency graph



Cloud9 IDE
Your code anywhere, anytime

github.com/c9/architect

Happy coding!



<http://c9.io>

Sergi Mansilla
@sergimansilla
github.com/sergi